IN THE UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF TEXAS
DALLAS DIVISION

SECURITYPROFILING, LLC, **Plaintiff/Counterclaim-Defendant,** **v.** **TREND MICRO AMERICA, INC. and TREND MICRO INCORPORATED,** **Defendants/Counterclaim-Plaintiffs.**	**Civil Action No. 3:17-cv-1484-N**

APPENDIX TO PLAINTIFF'S COMBINED OPENING AND RESPONSIVE CLAIM CONSTRUCTION BRIEF

Plaintiff SecurityProfiling, LLC hereby submits this Appendix to Plaintiff's Combined Opening and Responsive Claim Construction Brief.

Exhibit No.	Description	Appendix No.
A	Declaration of George Pazuniak in Support of Plaintiff SecurityProfiling LLC's Combined Opening and Responsive Claim Construction Brief	3-4
A-1	Data and Computer Security – Dictionary of standards concepts and terms	5-9
A-2	Prosecution history of '699 Patent -excerpt	10-11
A-3	US Patent No. 5,892,903	12-31
A-4	US Patent No. 6,298,445	32-50

A-5	US Patent No. 6,321,334	51-183
A-6	US Patent Application Publication No. 2002/0104014	184-215
A-7	US Patent Application Publication No. 2002/0026591	216-239
A-8	US Patent Application Publication No. 2002/0199122	240-252
A-9	US Patent Application Publication No. 2003/0014669	253-288

Dated: January 22, 2018

Respectfully submitted,

BUETHER JOE & CARPENTER, LLC

By: *<u>/s/ Christopher M. Joe</u>*
Christopher M. Joe
State Bar No. 00787770
Chris.Joe@BJCIPLaw.com
Michael D. Ricketts
State Bar No. 24079208
Mickey.Ricketts@BJCIPLaw.com

1700 Pacific Avenue
Suite 4750
Dallas, Texas 75201
Telephone: (214) 466-1272
Facsimile: (214) 635-1828

Of Counsel:

Sean T. O'Kelly (DE No. 4349)
George Pazuniak DE (No. 478)
Daniel P. Murray (DE No. 5785)
Thomas H. Kramer (DE No. 6171)
O'Kelly & Ernst, LLC
901 N. Market Street, Suite 1000
Wilmington, Delaware 19801
(302) 778-4000
(302) 295-2873 (facsimile)
sokelly@oeblegal.com
gp@del-iplaw.com
dmurray@oeblegal.com
tkramer@oelegal.com

Thomas F. Meagher
Alan Christopher Pattillo
Meagher Emanuel Laks Goldberg & Liao, LLP
One Palmer Square
Suite 325
Princeton, NJ 08542
(609) 454-3500
tmeagher@meagheremanuel.com
cpattillo@meagheremanuel.com

CERTIFICATE OF SERVICE

The undersigned certifies that the foregoing document was filed electronically in compliance with Local Rule 5.1(d). As such, this document was served on all counsel who are registered users of ECF on this 22nd day of January, 2018.

By: *<u>/s/ Christopher M. Joe</u>*

Christopher M. Joe

IN THE UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF TEXAS
DALLAS DIVISION

SECURITYPROFILING, LLC, **Plaintiff/Counterclaim-Defendant,** **v.** **TREND MICRO AMERICA, INC. and TREND MICRO INCORPORATED,** **Defendants/Counterclaim-Plaintiffs.**	**Civil Action No. 3:17-cv-1484-N**

DECLARATION OF GEORGE PAZUNIAK IN SUPPORT OF PLAINTIFF SECURITYPROFILING LLC'S COMBINED OPENING AND RESPONSIVE CLAIM CONSTRUCTION BRIEF

I, George Pazuniak, declare as follows:

1. My name is George Pazuniak. I am of counsel at O'Kelly, Ernst and Joyce, LLC (O'Kelly Ernst), counsel of record for SecurityProfiling, LLC in this action. I am admitted to practice before the Court in this matter. I have personal knowledge of the facts herein and, if called as a witness, could and would testify competently as to their truth.
2. Attached hereto as Exhibit A-1 Appendix 5-9 is a true and correct copy of the definition of "vulnerability" from the Data and Computer Security – Dictionary of standards concepts and terms, published by Macmillan Publishers Ltd. in 1989.
3. Attached hereto as Exhibit A-2 Appendix 10-11 is a true and correct excerpt of the Prosecution History of U.S. Patent No. 8,266,699, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.

4. Attached hereto as Exhibit A-3 Appendix 12-31 is a true and correct copy of US Patent No. 5,892,903, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.
5. Attached hereto as Exhibit A-4 Appendix 32-50 is a true and correct copy of US Patent No. 6,298,445, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.
6. Attached hereto as Exhibit A-5 Appendix 51-183 is a true and correct copy of US Patent No. 6,321,334, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.
7. Attached hereto as Exhibit A-6 Appendix 184-215 is a true and correct copy of US Patent Application Publication No. 2002/0104014, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.
8. Attached hereto as Exhibit A-7 Appendix 216-239 is a true and correct copy of US Patent Application Publication No. 2002/0026591, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.
9. Attached hereto as Exhibit A-8 Appendix 240-252 is a true and correct copy of US Patent Application Publication No. 2002/0199122, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.
10. Attached hereto as Exhibit A-9 Appendix 253-288 is a true and correct copy of US Patent Application Publication No. 2003/0014669, downloaded from https://portal.uspto.gov/pair/PublicPair on Jan. 20, 2018.

Dated: January 22, 2018

Respectfully submitted,

By: *<u>/s/ George Pazuniak</u>*
George Pazuniak

Exhibit A-1

DATA & COMPUTER SECURITY

Dictionary of standards concepts and terms

Dennis Longley
Michael Shain

Macmillan Reference Books

DATA & COMPUTER SECURITY

Dictionary of standards concepts and terms

Dennis Longley
Michael Shain

MACMILLAN
REFERENCE
BOOKS

First published in the United Kingdom by
MACMILLAN PUBLISHERS LTD (Journals Division), 1987

Reprinted 1988

Paperback first published 1989

British Library Cataloguing in Publication Data
Longley, Dennis
Data & computer security: dictionary of standards, concepts and terms.
1. Electronic data processing departments — Security measures — Dictionaries
I. Title II. Shain, Michael
005.8'03'21 HF5548.2

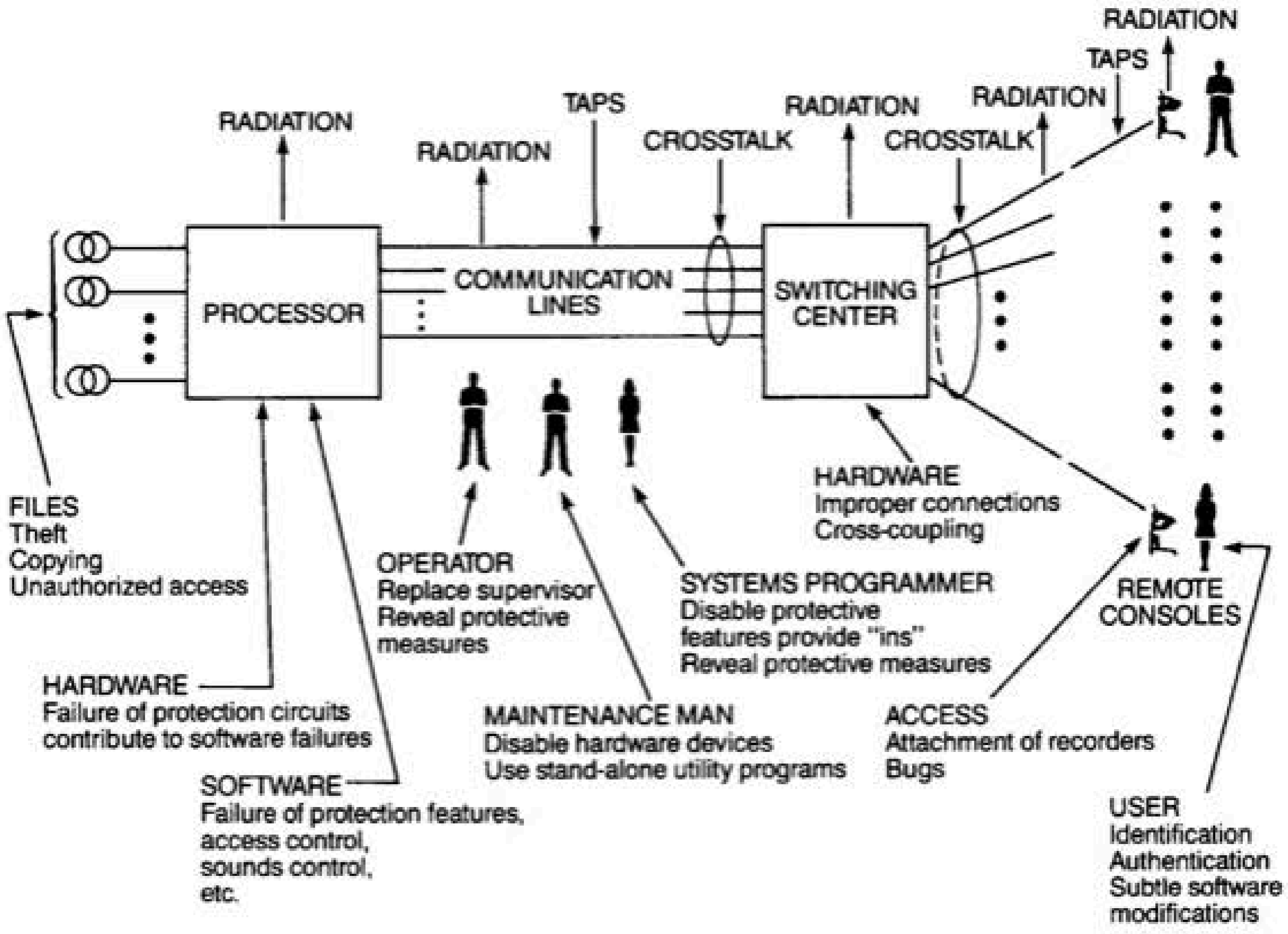
ISBN 978-0-333-42935-8
ISBN 978-0-333-51178-7 ISBN 978-1-349-11170-1 (eBook)
DOI 10.1007/978-1-349-11170-1

vostro account. *Synonymous with* LORO ACCOUNT.

V-series recommendations of CCITT. In data communications, a series of recommendations relating to data communications over analog channels. *Compare* X-SERIES RECOMMENDATIONS OF CCITT. *See* PROTOCOL STANDARDS.

vulnerability. (1) In computer security, a weakness in automated system security procedures, administrative controls, internal controls, etc. that could be exploited by a threat to gain unauthorized access to information or to disrupt critical processing. (AFR). (2) In computer security, a weakness in the physical layout, organization, procedures, personnel, management, administration, hardware or software that may be exploited to cause harm to the ADP system or activity. The presence of a vulnerability does not in itself cause harm; a vulnerability is merely a condition or set of conditions that may allow the ADP system or activity to be harmed by an attack. (OPNAVINST). *See* TECHNICAL VULNERABILITY. (3) In computer security, any weakness or flaw existing in a system. The susceptibility of a system to a specific threat attack or harmful event, or the opportunity available to a threat agent to mount that attack. *Compare* SAFEGUARD, THREAT. *See* HARMFUL EVENT, MATRIX METHODOLOGY, VULNERABILITY ASSESSMENT.

vulnerability assessment. (1) A review of the susceptibility to loss or unauthorized use of resources, errors in reports and information, illegal or unethical acts, and/or adverse or unfavorable public opinion. (OMBC). (2) A measurement of vulnerability which would include: (a) the susceptibility of a particular system to to a specific attack; (b) the opportunity available to a threat agent (methods or things that may be used to exploit a vulnerability, such as fire) to mount that attack. A vulnerability is always demonstrable, but may exist independently of a known threat. In general, a description of a



vulnerability
Some of the many possible threats to the security of a computer system. Because security threats arise from such a wide variety of sources, the mechanisms and procedures necessary to provide a secure environment must cover many areas of an enterprise.) Source: *Journal of Computers & Security.*

Exhibit A-2

O I P E
OCT 2 5 2004
PATENT & TRADEMARK OFFICE

309048 WEMMH/SB/08A (4/03)
Approved for use through 10/31/2002. OMB 0651-0031
U.S. Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number

Substitute for form 1449A/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(use as many sheets as necessary)

Sheet 1 of 1

Complete if Known	
Application Number	10/882,588
Filing Date	July 1, 2004
First Named Inventor	Brett M. OLIPHANT
Group Art Unit	2131
Examiner Name	
Attorney Docket Number	36029-4

U.S. PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Document Number Number-Kind Code[2] (If known)	Publication Date MM-DD-YYYY	Name of Patentee or Application of Cited Document	Pages, Columns, Lines where Relevant Passages or Relevant Figures Appear
SZ		US-5,335,346	8/2/1994	Fabbio	
SZ		US-5,765,153	6/9/1998	Benantar et al.	
SZ		US-5,892,903	4/6/1999	Klaus	
SZ		US-6,044,466	3/28/2000	Ananed et al.	
SZ		US-6,298,445	10/2/2001	Shostack et al.	
SZ		US-6,321,334	11/20/2001	Jerger et al.	
SZ		US-6,345,361	2/5/2002	Jerger et al.	
SZ		US-6,473,800	10/29/2002	Jerger et al.	
SZ		US-6,526,513	2/25/2003	Shrader et al.	
SZ		US-2001/034847	20/25/2001	Gaul, Jr.	
SZ		US-2002/026591	2/28/2002	Hartley et al.	
SZ		US-2002/104014	8/1/2002	Zobel et al.	
SZ		US-2002/112179	8/15/2002	Riordan et al.	
SZ		US-2002/199122	12/26/2002	Davis et al.	
SZ		US-2002/366525	12/20/2002	Toshio	
SZ		US-2003/005178	1/2/2003	Hemsath	
SZ		US-2003/014669	1/16/2003	Caceres et al.	
SZ		US-2003/037601	2/7/2003	Toshio	
SZ		US-2003/061506	3/27/2003	Cooper et al.	
SZ		US-2003/084320	5/1/2003	Tarquini et al.	

Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3]-Number[4]-Kind Code[5] (If known)	Publication Date MM-DD-YYYY	Name of Patentee or Application of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T[6]
SZ		WO 02/14987	2/21/2002	Gadish et al.		
SZ		WO 03/007192	1/23/2003	Caceres et al.		
SZ		WO 03/029940	4/10/2003	Singleton		
SZ		WO 03/029941	4/10/2003	Singleton		

Examiner Signature	Syed Zia	Date Considered	09/13/2007

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] Applicant's unique citation designation number (optional). [2] See Kinds Codes of USPTO Patent Documents at www.uspto.gov or MPEP 901.04. [3] Enter Office that issued the document, by the two-letter code (WIPO Standard ST.3). [4] For Japanese patent documents, the indication of the year of the reign of the emperor must precede the serial number of the patent document. [5] Kind of document by the apOpropriate symbols as indicated on the document under WIPO Standard ST.16 if possible. [6] Applicant is to place a check mark here if English language Translation is attached.

Burden Hour Statement: This form is estimated to take 2.0 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Exhibit A-3

US005892903A

United States Patent [19]

Klaus

[11] **Patent Number: 5,892,903**

[45] **Date of Patent: Apr. 6, 1999**

[54] **METHOD AND APPARATUS FOR DETECTING AND IDENTIFYING SECURITY VULNERABILITIES IN AN OPEN NETWORK COMPUTER COMMUNICATION SYSTEM**

[75] Inventor: **Christopher W. Klaus**, Atlanta, Ga.

[73] Assignee: **Internet Security Systems, Inc.**, Atlanta, Ga.

[21] Appl. No.: **710,162**

[22] Filed: **Sep. 12, 1996**

[51] **Int. Cl.**[6] **G06F 11/00**

[52] **U.S. Cl.** **395/187.01**; 395/200.57

[58] **Field of Search** 395/187.01, 186, 395/188.01, 200.59, 200.57, 183.04, 200.67, 200.68

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,223,380	9/1980	Antonaccio et al.	364/200
5,204,966	4/1993	Wittenberg et al.	395/188.01
5,309,562	5/1994	Li	395/200
5,311,593	5/1994	Carmi	380/23
5,347,450	9/1994	Nugent	395/200
5,371,852	12/1994	Attanasio et al.	395/200
5,515,508	5/1996	Pettus et al.	395/200.01
5,557,742	9/1996	Smaha et al.	395/186
5,623,601	4/1997	Vu	395/187.01

OTHER PUBLICATIONS

Guha et al., "Network Security via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposeed Solutions", IEEE, pp. 603–610, Mar. 1996.

Garg et al., "High Level Communication Primitives for Concurrent Systems", IEEE, pp. 92–99, 1988.

Hastings et al., "TCP/IP Spoofing Fundamentals", IEEE, pp. 218–224, May 1996.

Snapp, "Signature Analysis and Communication Issues in a Distributed Intrusion Detection System", Master Thesis; University of California, Davis, CA, pp. 1–40, 1991.

Guha et al., "Network Security via Reverse Engineering of TCP Code: Vulnerability Analysis and Proposed Solutions", IEEE, pp. 40–48, Jul. 1997.

Djahandari et al., "An MBone Proxy for an Application Gateway Firewall", IEEE, pp. 72–81, Nov. 1997.

Kim et al., "Implementing a Secure rlogin Environment: A Case Study of Using a Secure Network Layer Protocol", Department of Computer Science, University of Arizona, pp. 1–9, Jun. 1995.

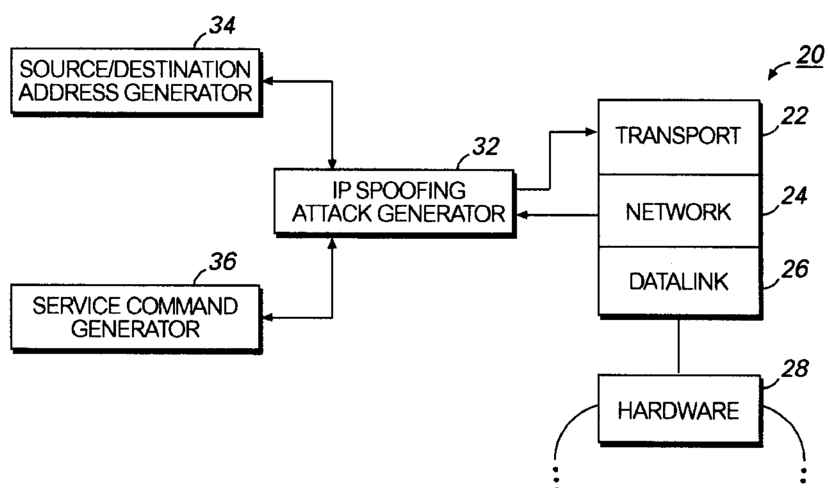
Satyanarayanan, "Integrating Security in a Large Distributed System", Acm Transactions on Computer Systems, vol. 7, No. 3, pp. 47–280, Aug. 1989.

Primary Examiner—Albert Decady
Assistant Examiner—Scott T. Badesman
Attorney, Agent, or Firm—Morris, Manning & Martin, L.L.P.

[57] **ABSTRACT**

A system and method is disclosed for detecting security vulnerabilities in a computer network. The system includes an IP spoofing attack detector, a stealth port service map generator, a source port verifier, source routing verifier, an RPC service detector and a Socks configuration verifier. Each of these verifiers may be operated separately or as a group to detect security vulnerabilities on a network. Each verifier may be programmed to exhaustively test all ports of all computers on a network to detect susceptibility to IP spoofing attacks, access to services with little or no authorization checks or misconfigured routers or Socks servers. The detected vulnerabilities or the location of services having little or no authorization checks may be stored in a table for reference by a network administrator. The service map generated by the stealth service map generator may be used to identify all service ports on a network to facilitate the operation of the other verifiers which send service command messages to service ports to detect their accessibility. A graphic user interface (GUI) may be used to provide input and control by a user to the security verifiers and to present options and display information to the user.

41 Claims, 8 Drawing Sheets



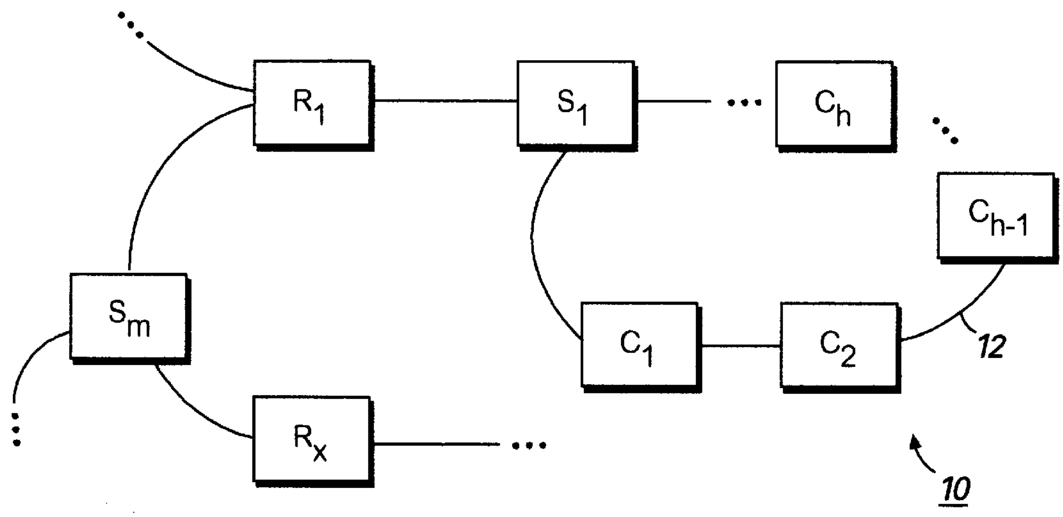

R1
S1
Ch
Ch-1
Sm
C1
C2
12
Rx
10

FIG.1

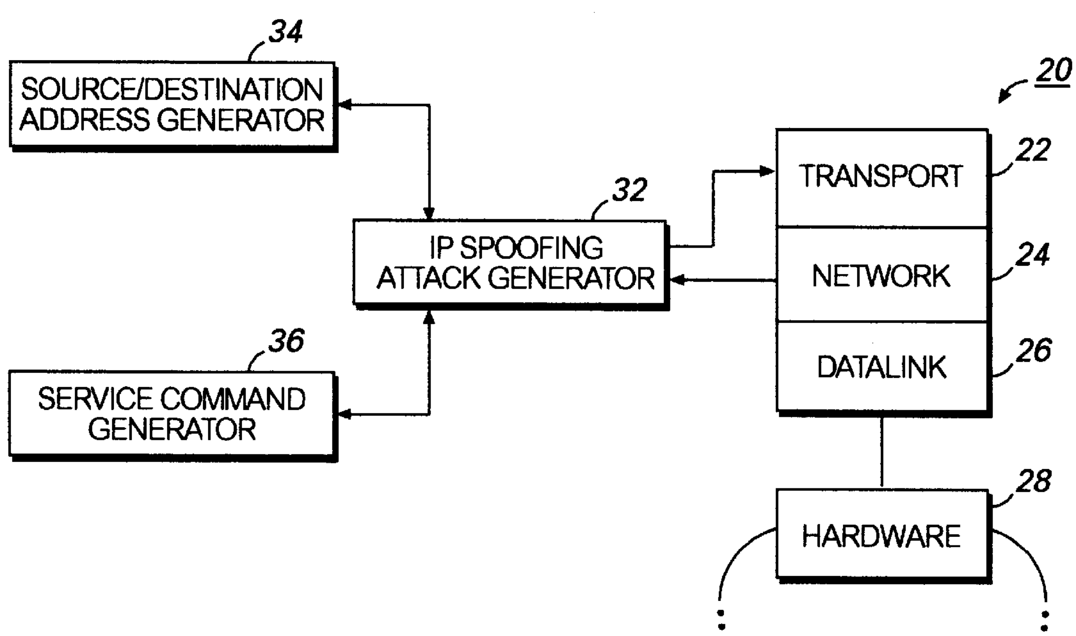
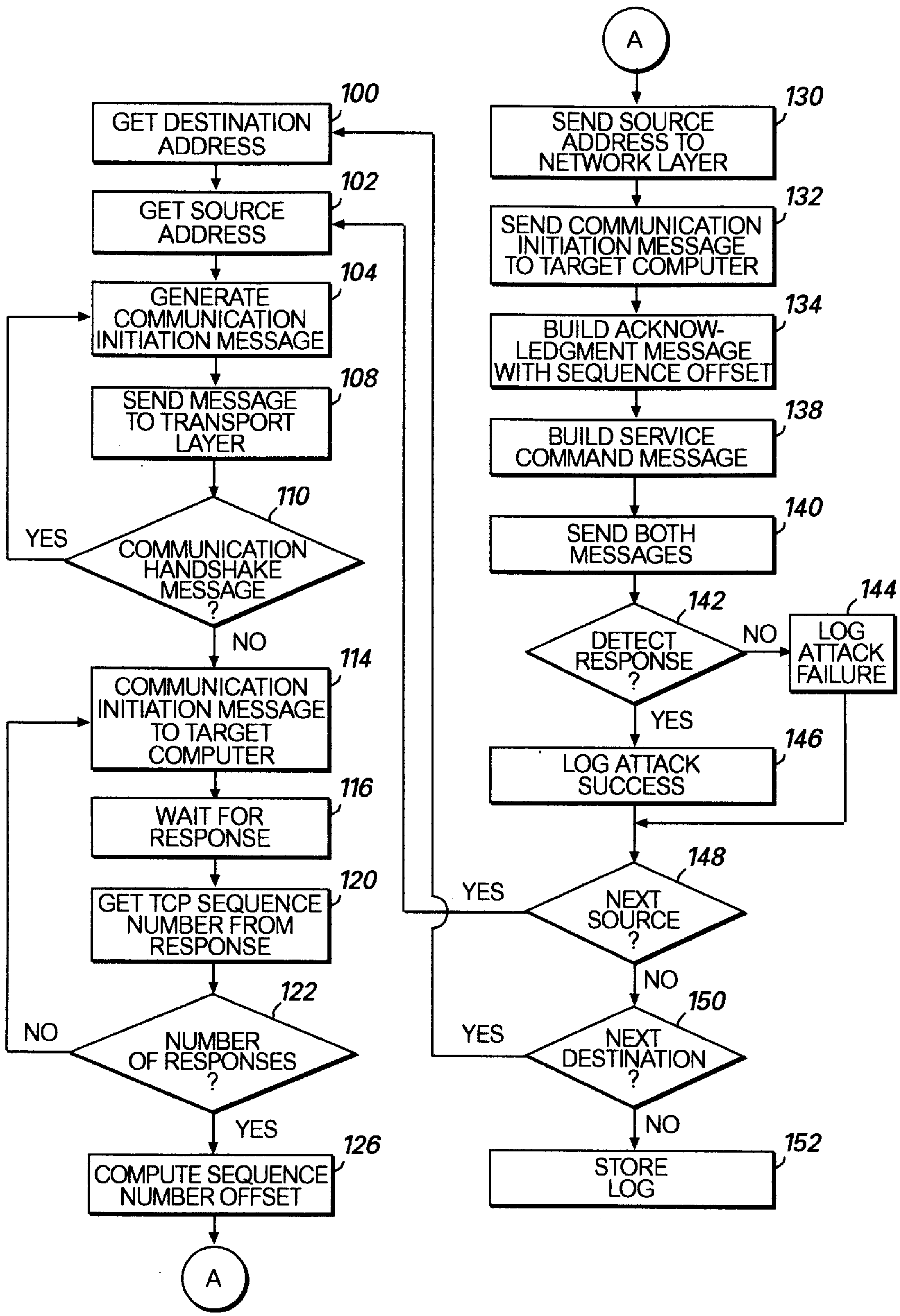
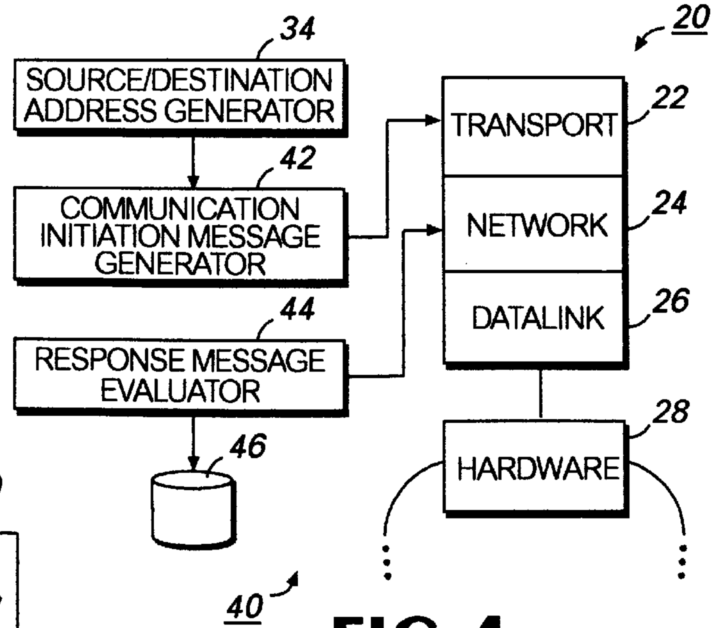

34
SOURCE/DESTINATION ADDRESS GENERATOR
20
22
TRANSPORT
32
IP SPOOFING ATTACK GENERATOR
NETWORK
24
DATALINK
26
36
SERVICE COMMAND GENERATOR
28
HARDWARE

FIG.2



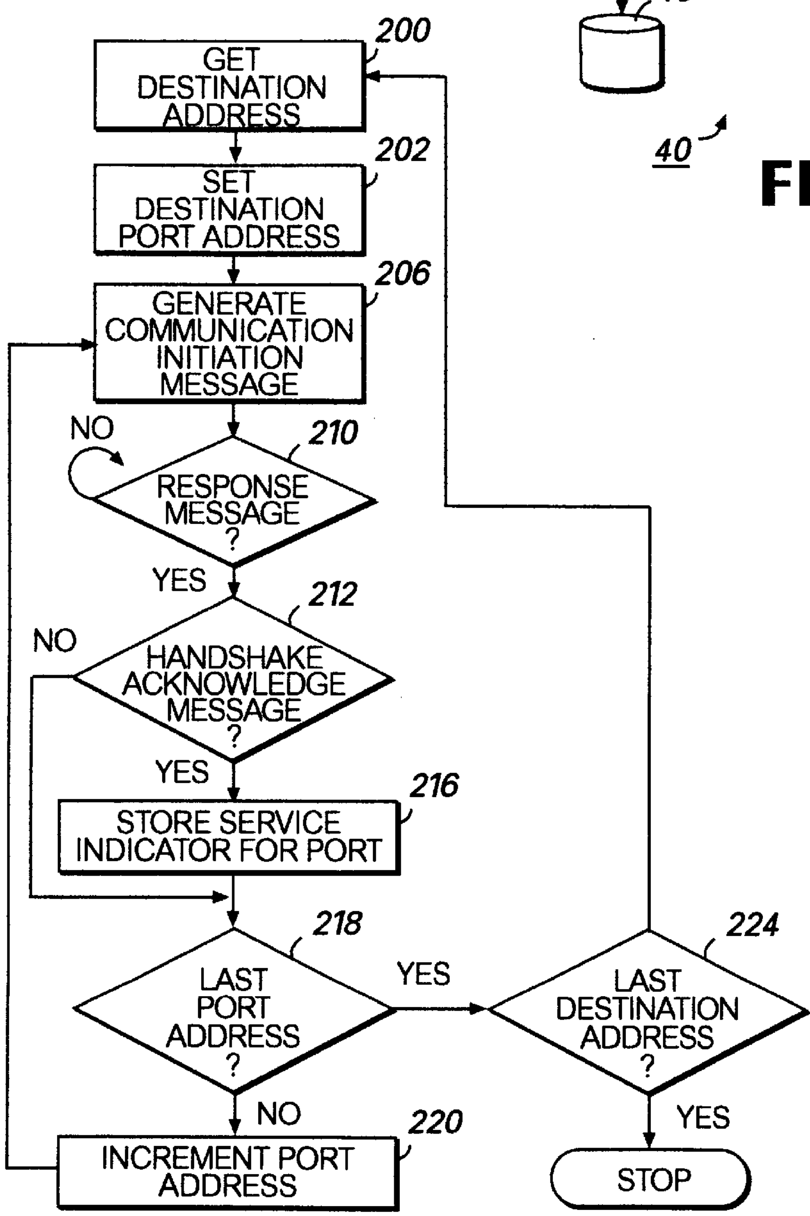
A
100
GET DESTINATION ADDRESS
102
GET SOURCE ADDRESS
104
GENERATE COMMUNICATION INITIATION MESSAGE
108
SEND MESSAGE TO TRANSPORT LAYER
110
COMMUNICATION HANDSHAKE MESSAGE ?
YES
NO
114
COMMUNICATION INITIATION MESSAGE TO TARGET COMPUTER
116
WAIT FOR RESPONSE
120
GET TCP SEQUENCE NUMBER FROM RESPONSE
122
NUMBER OF RESPONSES ?
NO
YES
126
COMPUTE SEQUENCE NUMBER OFFSET
A
130
SEND SOURCE ADDRESS TO NETWORK LAYER
132
SEND COMMUNICATION INITIATION MESSAGE TO TARGET COMPUTER
134
BUILD ACKNOW-LEDGMENT MESSAGE WITH SEQUENCE OFFSET
138
BUILD SERVICE COMMAND MESSAGE
140
SEND BOTH MESSAGES
142
DETECT RESPONSE ?
NO
144
LOG ATTACK FAILURE
YES
146
LOG ATTACK SUCCESS
148
NEXT SOURCE ?
YES
NO
150
NEXT DESTINATION ?
YES
NO
152
STORE LOG

FIG.3



20
34
SOURCE/DESTINATION ADDRESS GENERATOR
42
COMMUNICATION INITIATION MESSAGE GENERATOR
44
RESPONSE MESSAGE EVALUATOR
46
TRANSPORT
22
NETWORK
24
DATALINK
26
HARDWARE
28
40

FIG.4



GET DESTINATION ADDRESS
200
SET DESTINATION PORT ADDRESS
202
GENERATE COMMUNICATION INITIATION MESSAGE
206
NO
RESPONSE MESSAGE ?
210
YES
NO
HANDSHAKE ACKNOWLEDGE MESSAGE ?
212
YES
STORE SERVICE INDICATOR FOR PORT
216
LAST PORT ADDRESS ?
218
YES
LAST DESTINATION ADDRESS ?
224
NO
INCREMENT PORT ADDRESS
220
YES
STOP

FIG.5

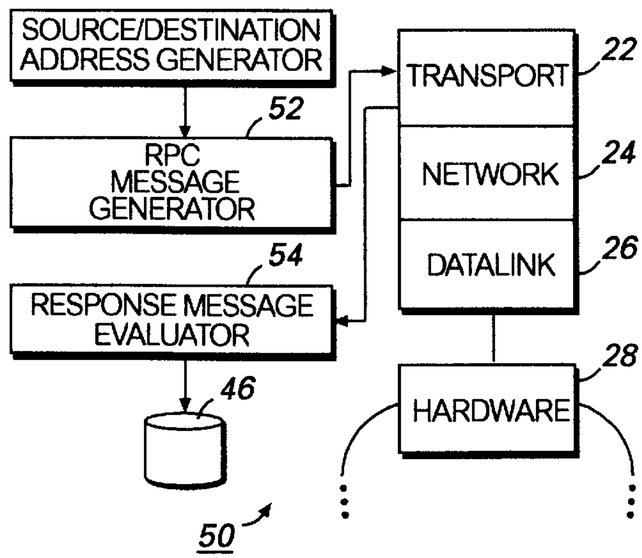

SOURCE/DESTINATION ADDRESS GENERATOR
52
RPC MESSAGE GENERATOR
54
RESPONSE MESSAGE EVALUATOR
46
TRANSPORT
22
NETWORK
24
DATALINK
26
HARDWARE
28
50

FIG.6

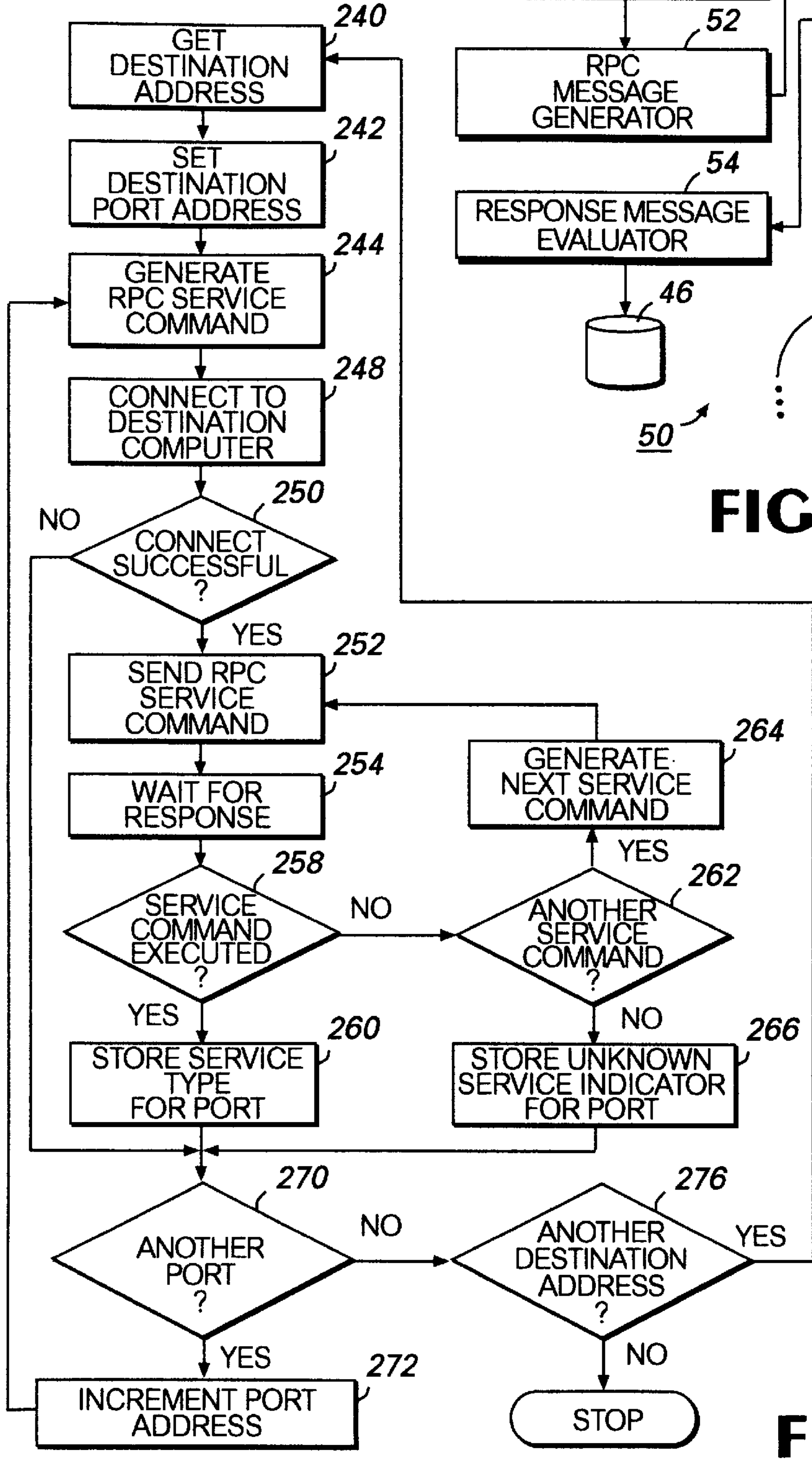

GET DESTINATION ADDRESS
240
SET DESTINATION PORT ADDRESS
242
GENERATE RPC SERVICE COMMAND
244
CONNECT TO DESTINATION COMPUTER
248
CONNECT SUCCESSFUL ?
250
NO
YES
SEND RPC SERVICE COMMAND
252
WAIT FOR RESPONSE
254
SERVICE COMMAND EXECUTED ?
258
NO
YES
GENERATE NEXT SERVICE COMMAND
264
ANOTHER SERVICE COMMAND ?
262
YES
NO
STORE SERVICE TYPE FOR PORT
260
STORE UNKNOWN SERVICE INDICATOR FOR PORT
266
ANOTHER PORT ?
270
NO
YES
INCREMENT PORT ADDRESS
272
ANOTHER DESTINATION ADDRESS ?
276
YES
NO
STOP

FIG.7

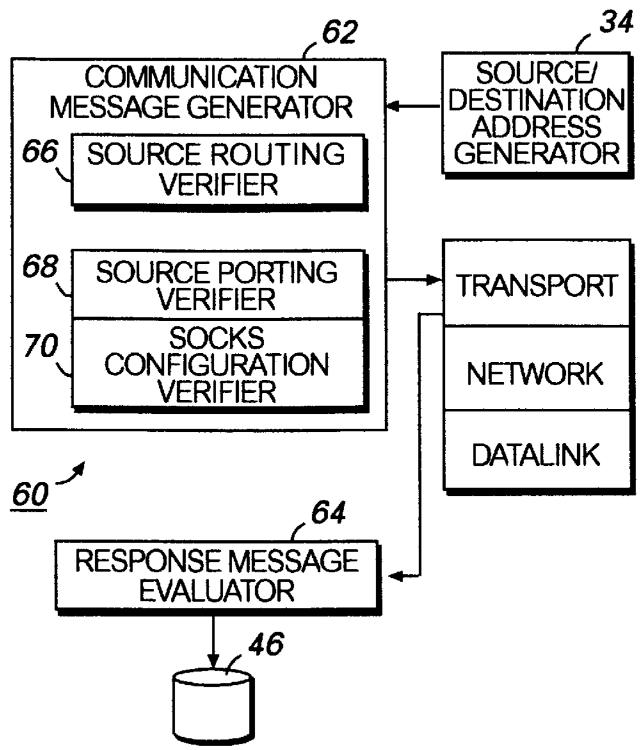

62
COMMUNICATION MESSAGE GENERATOR
66
SOURCE ROUTING VERIFIER
68
SOURCE PORTING VERIFIER
70
SOCKS CONFIGURATION VERIFIER
34
SOURCE/ DESTINATION ADDRESS GENERATOR
TRANSPORT
NETWORK
DATALINK
60
64
RESPONSE MESSAGE EVALUATOR
46

FIG.8

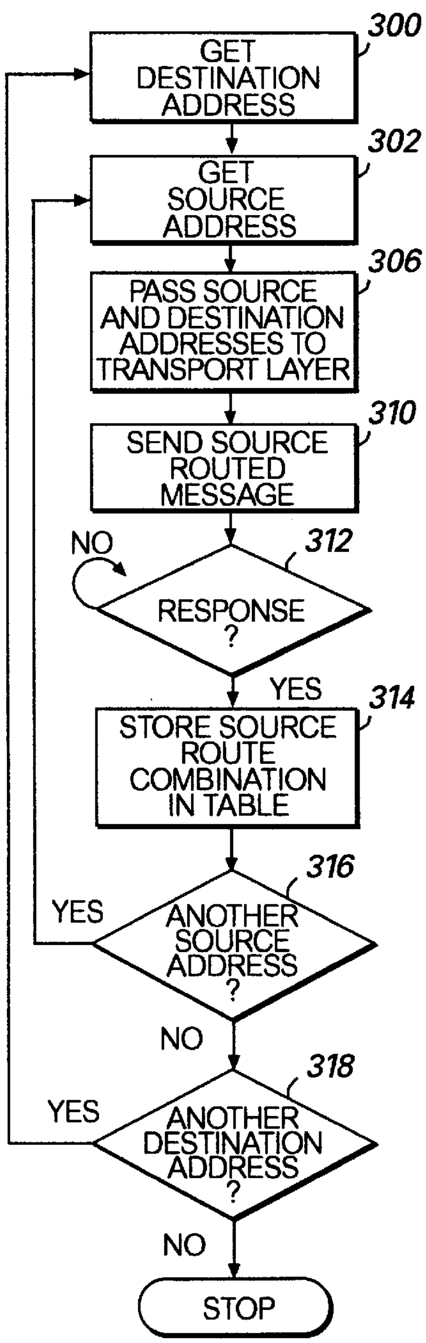

300
GET DESTINATION ADDRESS
302
GET SOURCE ADDRESS
306
PASS SOURCE AND DESTINATION ADDRESSES TO TRANSPORT LAYER
310
SEND SOURCE ROUTED MESSAGE
NO
312
RESPONSE ?
YES
314
STORE SOURCE ROUTE COMBINATION IN TABLE
316
YES
ANOTHER SOURCE ADDRESS ?
NO
318
YES
ANOTHER DESTINATION ADDRESS ?
NO
STOP

FIG.9

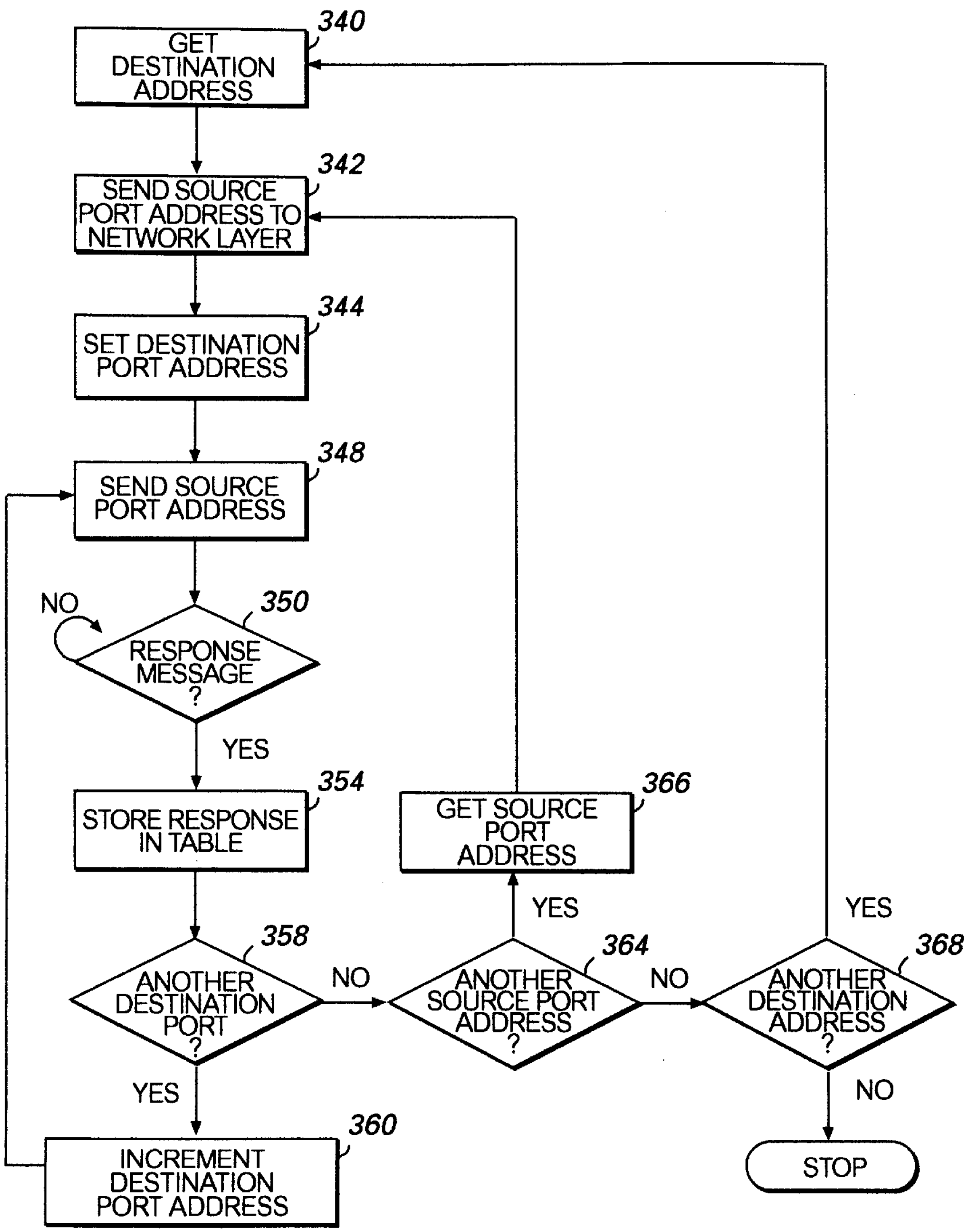

340
GET DESTINATION ADDRESS
342
SEND SOURCE PORT ADDRESS TO NETWORK LAYER
344
SET DESTINATION PORT ADDRESS
348
SEND SOURCE PORT ADDRESS
NO
350
RESPONSE MESSAGE ?
YES
354
STORE RESPONSE IN TABLE
358
ANOTHER DESTINATION PORT ?
NO
YES
360
INCREMENT DESTINATION PORT ADDRESS
366
GET SOURCE PORT ADDRESS
YES
364
ANOTHER SOURCE PORT ADDRESS ?
NO
YES
368
ANOTHER DESTINATION ADDRESS ?
NO
STOP

FIG.10

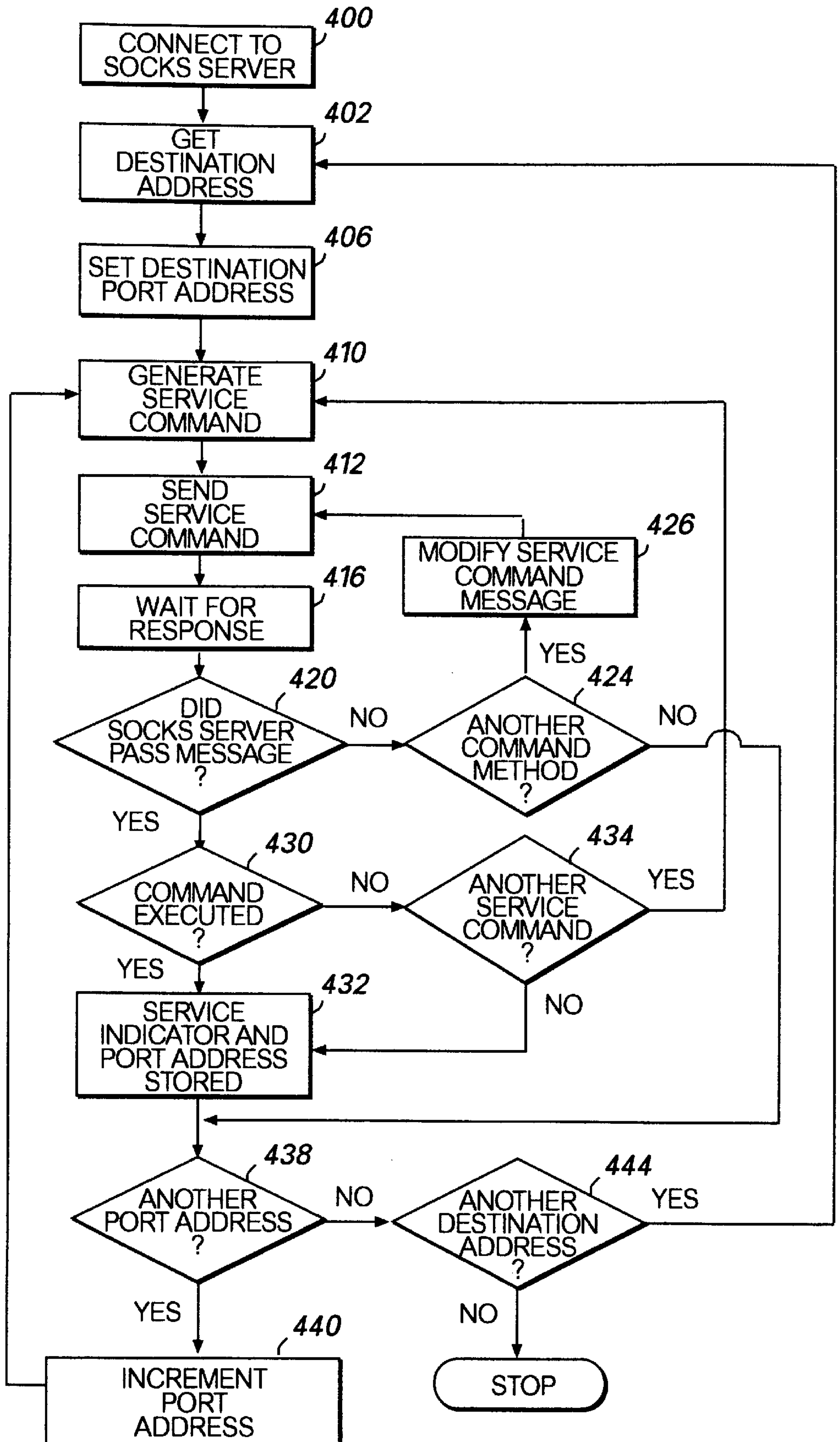

CONNECT TO SOCKS SERVER
400
GET DESTINATION ADDRESS
402
SET DESTINATION PORT ADDRESS
406
GENERATE SERVICE COMMAND
410
SEND SERVICE COMMAND
412
WAIT FOR RESPONSE
416
MODIFY SERVICE COMMAND MESSAGE
426
DID SOCKS SERVER PASS MESSAGE ?
420
NO
YES
ANOTHER COMMAND METHOD ?
424
YES
NO
COMMAND EXECUTED ?
430
NO
YES
ANOTHER SERVICE COMMAND ?
434
YES
NO
SERVICE INDICATOR AND PORT ADDRESS STORED
432
ANOTHER PORT ADDRESS ?
438
NO
YES
ANOTHER DESTINATION ADDRESS ?
444
YES
NO
INCREMENT PORT ADDRESS
440
STOP

FIG.11

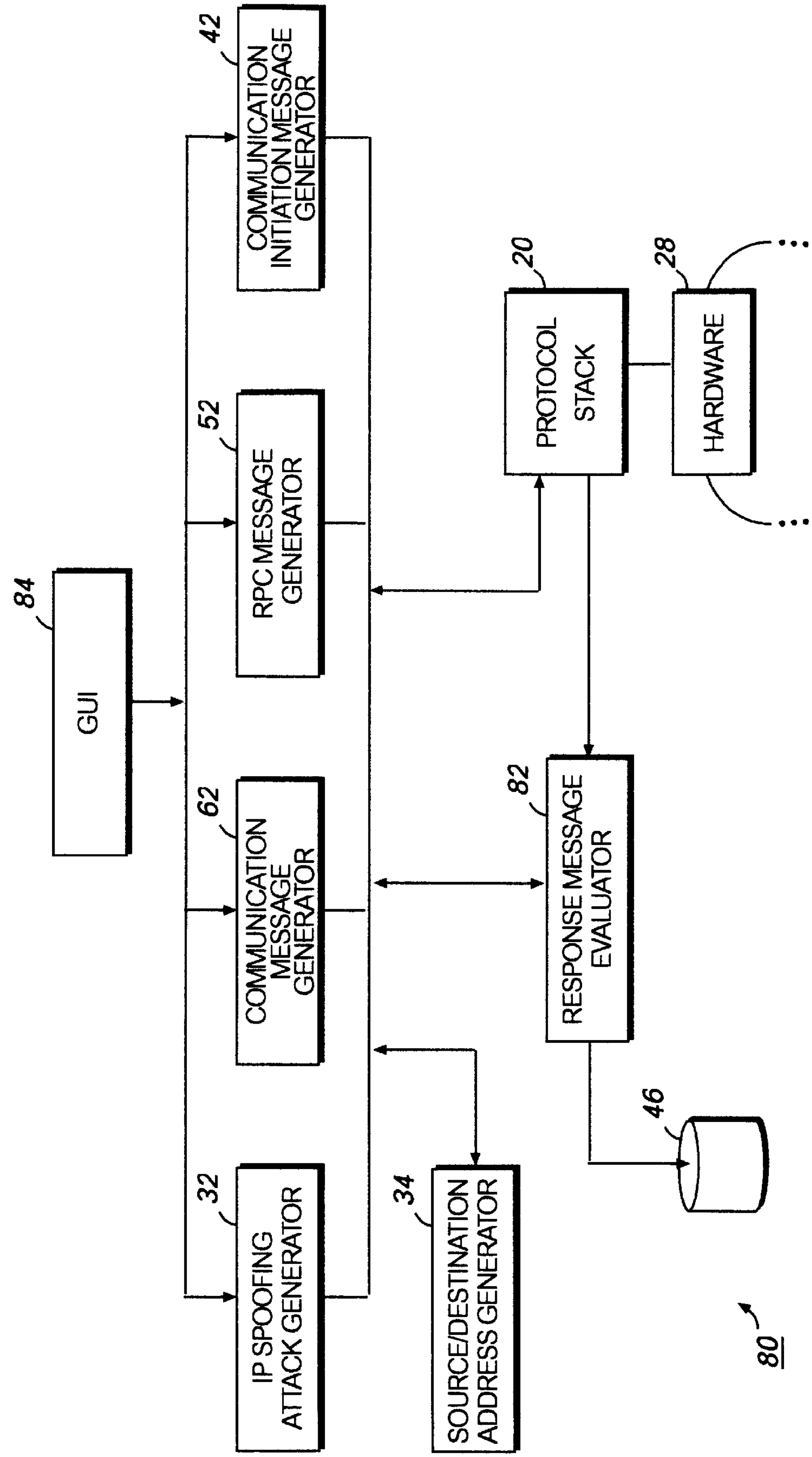

GUI
84
COMMUNICATION INITIATION MESSAGE GENERATOR
42
RPC MESSAGE GENERATOR
52
COMMUNICATION MESSAGE GENERATOR
62
IP SPOOFING ATTACK GENERATOR
32
SOURCE/DESTINATION ADDRESS GENERATOR
34
PROTOCOL STACK
20
HARDWARE
28
RESPONSE MESSAGE EVALUATOR
82
46
80

FIG.12

METHOD AND APPARATUS FOR DETECTING AND IDENTIFYING SECURITY VULNERABILITIES IN AN OPEN NETWORK COMPUTER COMMUNICATION SYSTEM

FIELD OF THE INVENTION

This invention relates to network communications for computers, and, more particularly, to computer communications over open networks.

BACKGROUND OF THE INVENTION

Many business and scientific organizations in the United States which use more than one computer in their operations couple the computers together through a network. The network permits the computers to be islands of processing which may share resources or data through communication over the network. The data which may be communicated over the network may take the form of programs developed on a user's computer, data files created on a user's computer, electronic mail messages and other data messages and files which may be generated or modified by a user at a user's computer. Typically, the user's computer includes an operating system for controlling the resources of the user's computer, including its central processing unit ("CPU"), memory (both volatile and non-volatile memory) and computer peripherals such as printers, modems and other known computer peripheral devices. The user typically executes application programs and system services to generate data files or programs.

Most computers are coupled to a network through a network communication printed circuit card which is typically resident within each computer system. This communication card typically includes processors, programs and memory to provide the electrical signals for transmission of data and implement the protocol which standardizes the messages transmitted through a network. To communicate data from a user's application program or operating system service, a protocol stack is typically implemented between the communication card for the network and the operating system services and application programs.

The typical protocol stack used on most open networks is a Transport Control Protocol/Internet Protocol ("TCP/IP"). This protocol stack includes a transport layer which divides a data stream from an application program or service into segments and which adds a header with a sequence number for each segment. The TCP segments generated by the transport layer are passed to the Internet Protocol ("IP") layer. The IP layer creates a packet having a packet header and a data portion. The data portion contains the TCP segment and the packet header contains a source address identifying the computer sending a message and a destination address identifying the computer for which the message is intended. The IP layer also determines the physical address of the destination computer or an intermediate computer, in some cases, which is intended to receive the transmitted message. The packet and the physical addresses are passed to a datalink layer. The datalink layer typically is part of the program implemented by a processor on the communication card and it encapsulates the packet from the IP layer in a datalink frame which is then transmitted by the hardware of the communication card. This datalink frame is typically called a packet. For purposes of this specification, the word "message" includes the data entities packet and datalink frame.

At the destination computer, the communication card implements the electrical specification of a hardware com-

munication standard, such as Ethernet, and captures a data message from a source computer. The datalink layer at the destination computer discards the datalink header and passes the encapsulated packet to the IP layer at the destination computer. The IP layer at the destination computer verifies that the packet was properly transmitted, usually by verifying a checksum for the packet. The IP layer then passes the encapsulated TCP segment to the transport layer at the destination computer. The transport layer verifies the checksum of the TCP message segment and the sequence number for the TCP packet. If the checksum and TCP sequence number are correct, data from the segment is passed to an application program or service at the destination computer.

Segregation of communication functions in the various layers of the protocol stack and the segregation of the protocol stack from the communication card and application programs, modularizes the functions required to implement communication over a computer network. This modularization of functions simplifies computer communication operation and maintenance. It also does not require a user to have knowledge of how the protocol stack and communication card communicate in order to send data messages to other computers over the network.

All of the computers coupled to a network may have approximately the same resources available at each machine. The type of network is sometimes called a peer to peer network. Another type of network environment is one in which one computer controls shared databases and other computer resources with other computers over the network. The computer controlling access to the shared resources is typically called a server and the computers utilizing the shared resources are called clients.

In both the client/server and peer to peer environments, a server or computer may be used as a gateway to other networks or computers. Another device which a message may encountered as it moves along a network is a router. A router examines destination addresses of messages it receives and routes them in an efficient manner to the specified destination computer. For example, a server on a first network may be coupled to a router which is coupled to a plurality of servers including a server on a second network and a server for a third network. In this type of environment, the computer on the first network may communicate with a computer on the third network by generating data messages which have the destination address for a computer on the third network. The message circulates through the first network and is eventually provided to the server of the first network. The server of the first network then passes the message to the router which determines that the message is addressed for the third network. Accordingly, it sends the message to the server of the third network. The communication facilities at the server for the third network recognize the destination address as existing on the third network and pass the message to a computer on the third network where it eventually would be passed to the destination computer.

While this type of communication effectively and efficiently couples all of the computers from all of the networks together without requiring a message to pass through each computer on the network, a message typically passes through a number of computers, routers, servers or gateways prior to reaching the destination computer. As a result, the data messages from one computer to another computer may be intercepted and data obtained from the message as the message is passed on to another computer. The type of network wherein this type of accessible communication is provided is typically called an open network. One of the more popularly known open networks is the Internet where

literally millions of servers and computers are coupled through a TCP/IP communication protocol.

While the open network architecture of the Internet permits a user on a network to have access to information on many different computers, it also provides access to messages generated by a user's computer and to the resources of the user's computer. In fact, there are persons who attempt to use knowledge regarding the operations of the protocol stack and operating systems in an effort to gain access to computers without authorization. These persons are typically called "hackers". Hackers present a significant security risk to any computer coupled to a network where a user for one computer may attempt to gain unauthorized access to resources on another computer of the network. For example, an employee may attempt to gain access to private and confidential employee records on a computer used by the human resources department of an employer.

In an effort to control access to a network and, hence, limit unauthorized access to computer resources available on that network, a number of computer communication security devices and techniques have been developed. One type of device which is used to control the transfer of data is typically called a "firewall". Firewalls are routers which use a set of rules to determine whether a data message should be permitted to pass into or out of a network before determining an efficient route for the message if the rules permit further transmission of the message. In this specification the term "routers" includes firewalls and routers.

In the TCP/IP protocol, a communication connection is established through a three handshake open network protocol. The first handshake or data message is from a source computer and is typically called a "synchronization" or "sync" message. In response to a sync message, the destination computer transmits a synchronization-acknowledgment ("sync-ack") message. The source computer then transmits an acknowledgment ("ack") message and a communication connection between the source and destination computer is established. To limit access to computers on a network, routers may be provided as a gateway to the network and programmed to detect and block sync messages being transmitted from a computer external to the network to a destination computer on the network. That is, computers on the network may send out sync messages through the router to initiate communication with other computers, but computers outside the router and its network cannot send sync messages through the router to initiate communication with computers on the network. In this way, a hacker cannot attempt to initiate communication with a computer on the network.

Hackers, however, have developed other ways which may be helpful in bypassing the screening function of a router. For example, one computer, such as a server on the network, may be permitted to receive sync messages from a computer outside the network. In an effort to get a message to another computer on a network, a hacker may attempt to use source routing to send a message from the server to another computer on the network. Source routing is a technique by which a source computer may specify an intermediate computer on the path for a message to be transmitted to a destination computer. In this way, the hacker may be able to establish a communication connection with a server through a router and thereafter send a message to another computer on the network by specifying the server as an intermediate computer for the message to the other computer.

In an effort to prevent source routing techniques from being used by hackers, some routers may be configured to intercept and discard all source routed messages to a network. For a router configured with source routing blocking, the router may have a set of rules for inbound messages, a set of rules for outbound messages and a set of rules for source routing messages. When a message which originated from outside the network is received by such a router, the router determines if it is a source routed message. If it is, the router blocks the message if the source routing blocking rule is activated. If blocking is not activated, it allows the source routed message through to the network. If the message is not a source routed message, the router evaluates the parameters of the message in view of the rules for receiving messages from sources external to the network. One such rule is the external sync message filter discussed above. Other rules may also be implemented in such a router. However, a router vulnerability exists where the rules used by the router are only compared to messages that are not source routed and the source routed blocking rule is not activated. In this situation, the router permits source routed messages through without comparing them to the filtering rules. In such a case, a computer external of the network may be able to bypass the external sync message filter and establish a communication connection with a computer on the network by using source routed messages.

What is needed is a system and method for verifying that the source routing blocking feature of a router has been activated.

Networks may also be coupled to external computers through a specialized communication filter typically known as a "Socks" proxy server. A Socks proxy server is interposed between a network and external computers. For an external computer to establish communication with a computer on a network coupled to a Socks server, the external computer first establishes a communication connection with the Socks server and the Socks server establishes a communication connection with the destination computer. Thereafter, the Socks server relays messages between the external computer and a computer on the network only if they comply with the filter rules configured for the Socks server. Typically, Socks servers are used to interface e-mail, File Transfer Protocol ("FTP") and Telnet communication services between computers on a network and computers external of the network and to block access to most other ports on a network. The interrogation and evaluation of messages through a Socks server is dependent upon the network administrator for proper configuration. Known methods for verifying the configuration of the Socks server is to view the configuration files of the Socks server to verify the rules are properly set. However, this method does not ascertain the rules actually being implemented by the Socks server.

What is needed is a method and system for determining the rules being implemented by a Socks server without reviewing the configuration files for a Socks server.

Another entry port for hackers are commonly known services which provide information to external users without requiring authorization checks such as passwords. Most implementations of the UNIX operating system, for example, include Remote Procedure Call (RPC) services which may not be protected by authorization checks. The ports on which RPC services are located may be determined by querying a UNIX operating system service known as "portmapper". In an effort to obtain knowledge regarding accessible services on a computer, a hacker may make an inquiry of the portmapper service at its port in order to obtain information regarding the RPC services available for entry on the computer. Although the portmapper service may

be reconfigured to include an authorization check that still does not provide an authorization check for the RPC services themselves.

What is needed is a system and method for detecting and reporting to a network administrator those ports which are coupled to RPC services which have little or no authorization checks.

As discussed above, the transport layer of the protocol stack provides a sequence number for each data segment to be transmitted. In the TCP/IP protocol, the sequence number is called a TCP sequence number which is placed in the TCP header generated by the transport layer. The sequence number for the data segment is typically incremented at predefined time units, for example, each second, and for each communication connection or attempted communication connection. For example, in attempting to establish communication with another computer on a TCP/IP network, the source computer generates a sync message with a TCP sequence number. The destination computer responds with a sync/ack message where the ack value in the message is the sequence number from the received sync message and the sequence number for the destination computer is a number generated by the destination computer. This sequence number typically has the value of the last TCP sequence number generated by the destination computer plus the addition of a preferred offset value for each predefined time unit and communication connection that has occurred since the last TCP sequence number was generated. The ack message from the source computer to the destination computer which completes the communication connection must include the TCP sequence number received from the destination computer in the sync/ack message.

One known way which hackers attempt to access a computer on a network is to emulate the communication of messages from another computer on the network. A hacker emulates another computer on the network by first blocking a communication port on the computer being emulated by repeatedly sending sync messages to a port on the computer. This causes the communication program for the port to fill its communication buffer with half-open communication connections. When the buffer is full, no more sync messages are accepted until the oldest attempted half-open communication connection times out. Typically, the time out period is ten minutes or longer. In order to obtain a sequence number, the hacker's computer sends a number of sync messages to the computer which is the target of the attack which responds with a plurality of sync/ack messages containing TCP sequence numbers to the hacker's computer. The TCP sequence numbers from the sync/ack messages may be compared to statistically determine the offset used by the target computer to generate TCP sequence numbers. The hacker then uses the emulated computer's blocked port address as the source computer address for a sync message originated by the hacker's computer. In response, the target computer replies with a sync/ack message which is addressed to the blocked computer port of the emulated computer. Thus, the hacker's computer does not receive the sync/ack message with the TCP sequence number required for a proper response. However, the hacker's computer then sends an ack message with the next computed sequence number derived from bombarding the target computer with sync messages. If the sequence number has been correctly computed so that it matches the sequence number in the sync/ack message sent by the target computer to the blocked computer port, a communication connection is established and the hacker is able to transmit a command to the service on the port of the target computer through which communication has been established. In a UNIX system, a hacker normally attacks the ports coupled to the rsh and rlogin services since the authorization check for these services is usually the source address. If the hacker is able to successfully emulate a computer on the network having an address authorized for the service on the target computer port, the command is executed by the service. The service command typically provided to the port of the target computer disrupts the target computer's operation so the hacker's computer has unencumbered access to the target computer's resources. These types of attacks which use predicted TCP sequence numbers are typically known as IP spoofing attacks.

Although the protocol stack for each computer uses different offset values to generate the initial TCP sequence number for establishing communication links, some machines generate initial sequence numbers which are more easily predicted than others. What is needed is a way of detecting which computers on a network are susceptible to attacks using predicted TCP sequence numbers.

SUMMARY OF THE INVENTION

The above-noted vulnerabilities of a computer network may be automatically detected by a computer program which implements the system and method of the present invention. One embodiment of the present invention includes an Internet protocol ("IP") spoofing attack generator for generating an IP spoofing attack directed to a target computer and a service command message generator for sending a command to be executed by a service coupled to a port on the target computer so that in response to the target computer being compromised by the IP spoofing attack the target computer generates a compromise indicator without altering or destroying the target computer's services and/or operations. Preferably, the target computer response is an electronic mail message or a Telnet initiation message. Preferably, the IP spoofing attack is directed against a port coupled to the rsh or rlogin services. Preferably, the embodiment includes a source/destination address generator which generates source and destination addresses for messages corresponding to an open network protocol. The destination addresses correspond to the target computer and the source addresses correspond to the emulated computer in the IP spoofing attack. The source/destination address generator generates the address for each computer on a network so that an IP spoofing attack from every computer on the network is directed against each of the other computers on the network. In this manner, those computers on the network which are most susceptible to an IP spoofing attack may be detected and modification of the TCP sequence number generator in the protocol stack may be adjusted to make an IP spoofing attack less likely to succeed.

Another embodiment of the present invention for detecting security vulnerabilities in the configuration rules of a router includes a communication message generator for generating and sending communication messages to computers coupled through an open network to a router and a response message detector for detecting responses from computers on the network generated in response to the communication messages. This embodiment of the present invention detects the vulnerability of the router to pass communication messages to computers on the network. Depending on the type of communication or service command message to which a computer responds, the inventive system may determine rules not implemented by a router. In one preferred embodiment, the communication message generator includes a Socks configuration verifier which establishes a communication connection with a Socks server

and attempts to send service command messages for different services with source addresses for computers on the network. The responses of the destination computer are examined to determine the types of messages which the Socks server passes to computers on the network from computers external to the network. This system may be used to verify the rules actually implemented by a Socks server.

In another embodiment, the communication message generator includes a source porting verifier which sets the source port address in a header for a generated communication message to a predetermined value to see if the router passes externally generated messages having the specified source port address to the network. Preferably, the predetermined value is the default source port identifier for a service having a known required predetermined source port address such as an FTP service. In this manner, the system of the present invention detects whether a computer external of the network can establish a communication connection with a computer on the network by using a predetermined source port identifier to avoid other rules in a router.

In another embodiment of the present invention, the communication message generator includes a source routing verifier which generates source-routed communication messages to determine whether the router has a source router message blocking rule activated. This embodiment may be used to determine whether the rules that the router applies to communication messages originated by computers external to the network may be bypassed by using source routed messages.

In another embodiment of the present invention, an RPC message generator generates RPC service command messages which are sent to ports of computers on a network to detect the ports coupled to RPC services having little or no authorization checks. These ports and the coupled services, if determined, may be stored and provided to a network administrator for installing more rigorous authorization checks.

In another embodiment of the present system, a communication initiation message generator for generating communication initiation messages for a three handshake protocol and a response message evaluator are used to determine which of the ports on each computer in a network have a service coupled thereto. This inventive system operates by sending sync messages to each port on every computer on the network and building a table of service identifiers which identify those ports which responded with a message indicating the presence of a service. Preferably, the communication initiation message is a sync message for TCP/IP networks and the messages indicating a service is coupled to a port is a sync/ack message. In this manner, the inventive system may build a map of those ports of each computer on the network which have service coupled thereto without creating a log of any communication connections on any the computers on the network. Since communication connections are only established and logged when the originating computer sends the ack message, this embodiment generates a map of available services in a stealth manner. This embodiment of the inventive system may be coupled with one or more of the other embodiments which generate service command messages to eliminate ports from the attempts to detect vulnerable services. Such a system speeds the security analysis of a network.

These and other advantages and benefits of the present invention may be ascertained from reading of the detailed specification in conjunction with the drawings.

DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated and constitute a part of this specification, illustrate a number of embodiments of the invention and, together with the general description given above, and the detailed description of the embodiments given below, serve to explain the principles of the invention.

FIG. **1** is a schematic of an open network system;

FIG. **2** is a block diagram of an embodiment of the present invention used to detect IP spoofing attack vulnerability;

FIG. **3** is a flow chart of the preferred process implemented by the embodiment depicted in FIG. **2**;

FIG. **4** is a block diagram of an embodiment of the present invention used to map the ports of computers of a network which are coupled to services without generating communication connections;

FIG. **5** is a flow chart of the preferred process implemented by the embodiment depicted in FIG. **4**;

FIG. **6** is a block diagram of an embodiment of the present invention used to detect Remote Procedure Call (RPC) services available on a network which have little or no authorization checks;

FIG. **7** is a flow chart of the preferred process implemented by the embodiment shown in FIG. **6**;

FIG. **8** is a block diagram of an embodiment of the present invention used to verify the configuration of routers and/or Socks servers;

FIG. **9** is a flow chart of the preferred process implemented by the source routing verifier of FIG. **8**;

FIG. **10** is a flow chart of the preferred process implemented by the source porting verifier of FIG. **8**;

FIG. **11** is a flow chart of the preferred process implemented by the Socks server verifier of FIG. **8**; and

FIG. **12** is a block diagram of a preferred embodiment of the present invention which incorporates the components of the systems shown in FIGS. **2**, **4**, **6** and **8**.

DETAILED SPECIFICATION OF EMBODIMENTS OF THE INVENTION

An open network system in which a system made in accordance with the principles of the present invention may be used is shown in FIG. **1**. An internetwork **10** may be comprised of a network **12** which in turn may be coupled to other servers, gateways and routers. Network **12** includes a plurality of computers C_1–C_n which are coupled through network **12** to a server S_1. This network in turn may be coupled to a router R_1 to provide further secured computer communication with other servers represented by S_m or other routers labeled R_x as shown in FIG. **1**. Although the principles of the present invention are extensible to other protocols, the invention is preferably used on networks which utilize the TCP/IP protocol. The computer program implementing a system or method of the present invention may reside on any of the computers on the network **12** or any server or any router of internetwork **10**.

Structure of a system embodiment made in accordance with the principles of the present invention is shown in FIG. **2**. A computer executing a program implementing the system or method of the present invention would typically include the programs and communication hardware card which implement a protocol stack **20**. Protocol stack **20** is comprised of transport layer **22**, network layer **24** and datalink layer **26**. These layers of protocol stack **20** operate in the well-known manner set forth above. The data frame prepared by datalink layer **26** is passed to communication hardware **28** for transmission to other computers in accordance with the source and destination information provided in the various headers generated by protocol stack **20**.

In one embodiment of the present invention which detects a computer's vulnerability to IP spoofing, the system includes an IP spoofing attack generator **32**, a source/destination address generator **34** and a service command generator **36**. Source/destination address generator **34** identifies the internet and physical addresses of the computers on the network **12** to be tested. Source/destination address generator **34** verifies that each computer on network **12** is emulated in IP spoofing attacks on all of the other computers on network **12**. In this manner, the inventive system exhaustibly tests all possible attack combinations on a network. Service command generator **36** generates commands for a service which may be coupled to a port which IP spoofing attack generator **32** is able to initiate a communications connection. Preferably, service command generator **36** generates commands for services which have little or no authorization checks. "Little" means that the authorization check verifies a computer address is on the network **12** or the like while "no" authorization check means the service executes any valid server command received on a port regardless of originating source. Preferably, service commands are generated for electronic mail, file transport protocol (FTP) and Telnet services. These commands preferably indicate that a target computer identified by a destination address has been compromised without altering the target computer's operational parameters such as changing system privileges for a user or deleting data files. Examples of such commands include a Telnet session initiation command such as telnet attack_computer_address where attack_computer_address is the address of the computer which performed the IP spoofing attack on the target computer. Another example of such a message is mail admin message where admin indicates the system or network administrator's mailbox and message indicates the contents of the message informing the administrator of the compromise. The service command received from command message generator **36** and the source and destination addresses received from source/destination address generator **34** are used by IP spoofing attack generator **32** to provide data and header content for messages sent to transport layer **22** and network layer **24** of protocol stack **20** which are used to implement the IP spoofing attack and detection.

The process implemented by IP spoofing attack generator **36** is shown in FIG. **3**. That process begins by obtaining a destination address (Block **100**) and a source address (Block **102**) from source/destination address generator **34**. Attack generator **32** then generates a communication initiation message for a three handshake protocol which is preferably a synchronization or sync message for the TCP/IP protocol (Block **104**). The communication initiation message is sent to a port on the source address computer by placing the message in a TCP segment and passing it to the transport layer (Block **108**). Transport layer **22**, network layer **24** and datalink layer **26** all appropriately encapsulate the sync message for transmission to the computer at the source address which is the address of the computer to be emulated in the IP spoofing attack. The process awaits the reception of a handshake acknowledgment message from the computer at the source address (Block **110**). The handshake acknowledgment message in the TCP/IP protocol is a sync/ack message. If a sync/ack message is received, another sync message is generated and sent to the same port address of the computer at the source address. This process continues until no sync/ack message is received from the computer at the source address within a predetermined time. These steps are performed to fill the communication buffer for a port on the source address computer with half-opened communication connections. This full buffer condition exists until the time period for completing a communication connection expires. In most computers, the expiration period is at least 10 minutes which is typically enough time to complete the attack. Because its buffer is full, this port on the computer at the source address no longer responds to communication initiation messages.

A sync message is then generated and transmitted to the computer at the destination address which now defines the target computer (Block **114**). The process waits for a sync/ack message from the computer at the destination address (Block **116**). When it is received, the process retrieves the TCP sequence number from the TCP segment header (Block **120**) and checks to see if a predetermined number of TCP sequence numbers have been retrieved from the target computer at the destination address (Block **122**). If the predetermined number of sequence numbers has not been received, a time period corresponding to the unit of time between changes in TCP sequence number modifications is delayed. This delay permits the computer at the destination address to modify the TCP sequence number which is used for initiating a communication session. Alternatively, the destination port address on the target computer may be changed to cause a sequence number increment as well. After this delay has expired or the destination port address changed, another sync message is generated and sent to the target computer (Block **114**). When the predetermined number of TCP sequence numbers have been received, the TCP numbers are used to evaluate the offset between TCP sequence numbers or the pattern for generating the TCP numbers (Block **126**). For example, if a predetermined offset amount is added to generate a new TCP sequence number for communication initiation, three TCP sequence numbers may be used to compute the difference between two adjacent TCP numbers. This difference should indicate the predetermined offset so that the next TCP sequence number which would be used by the target computer to respond to a new sync message is determined.

The IP spoofing attack process continues by setting the source address in the network layer **24** to the source address retrieved from source/destination address generator **34** (Block **130**). Now messages generated by the computer implementing the system and method of the present invention generates messages which appear to be originated from the computer at the source address. A communication initiation message is then generated and transmitted to the computer at the destination address (Block **132**). A period of time is delayed which corresponds to the normal response time for the target computer to send a sync/ack message. The process then prepares an ack message with the predicted TCP sequence number (Block **134**). A service command is obtained from a service command generator **36** and placed in a TCP segment passed to transport layer **22** to build a service command message (Block **138**). Both messages are then transmitted to the target computer to emulate an ack message and service command message from the emulated computer with the blocked port. If the predicted TCP sequence number for the ack message having the source address of the emulated computer matches the TCP sequence number sent by the target computer in the sync/ack message, the target computer establishes a communication connection which accepts messages having a source address of the emulated computer. Now the service command message sent from the computer implementing the process of FIG. **3** is accepted and executed by the service coupled to the port if the command is valid for the service. Preferably, the service command causes the computer at the destination

address to log the attack at the computer which has been compromised and, most preferably, the command causes the target computer to send a compromise indicator to the computer implementing the process of FIG. **3**, although another computer may receive the compromise indicator. The success or failure of the attack is logged (Block **142–146**). Preferably, a Telnet session is established between the compromised target computer and the computer executing the program which implements the process of FIG. **3**. Initiation of the Telnet session may be logged to record the success of the IP spoofing attack and additional information may be obtained during the Telnet session about the compromised computer to search for other security vulnerabilities of the target system.

The process then determines whether another source address exists on the network (Block **148**), and if there is, an attack on the target computer is attempted using the computer at the new source address as the emulated computer. If all of the source addresses have been used, the process checks to see if another destination address is available (Block **150**). If another source address is available, the process is repeated to evaluate attacks from each of the other computers on the network on the target computer defined by the new destination address. This process continues until each computer on the network has been used to attack all the other computers on the network. Once this has been done, the attack log may be stored in table **46**. The log may be later displayed to identify those computers on the network that are susceptible to IP spoofing attacks or provide other information obtained from the target computers that were compromised (Block **152**).

Another embodiment of the present invention is shown in FIG. **4**. System **40** includes a communication initiation message generator **42** and a response message evaluator **44** for determining whether a service is coupled to a port responding to a communication initiation message. System **40** builds a topology table **46** of service ports for network **12** from the communication initiation responses without causing a communication connection which may be logged by the computer having the ports which are being interrogated. Communication initiation message generator **42** is coupled to transport layer **22** of protocol stack **20** so communication initiation messages may be provided to transport layer **22** for transmission to the ports of the other computers coupled to network **12**. Preferably, the communication initiation messages are sync messages used in the three handshake protocol of a TCP/IP network. Response evaluator **44** is also coupled to transport layer **22** to receive the response messages to the communication initiation messages sent by a computer executing a program implementing the process shown in FIG. **5**. If the response message is the handshake acknowledgment message in the communication connection process, response evaluator **44** records the port address as a service access port for network **12** in table **46**. In the three handshake protocol used to establish a communication connection on a TCP/IP network, a sync/ack message is the handshake acknowledgment message which indicates a service is present on a port.

The process implemented by system **40** of FIG. **4** is shown in FIG. **5**. The process begins with communication initiation message generator **42** obtaining a destination address of a computer on network **12** from source/destination address generator **34** (Block **200**) and the destination port address is set to the first port address on the destination computer (Block **202**). Most computers in a TCP/IP protocol have port addresses in the range of 0–65,535. Preferably, each port address is tested by system **40**. A

communication initiation message is generated for the first port address of the computer at the destination address and passed to transport layer **22** (Block **206**). After the communication initiation message is transmitted, response evaluator **44** waits for receipt of a response message from the port to which the communication initiation message was sent (Block **210**). Response evaluator **44** then determines whether the message is a handshake acknowledgment message (Block **212**). If it is, response evaluator **44** stores a service indicator, the destination address and port address in service topology table (Block **216**). In a TCP/IP network, a sync/ack message indicates a service is coupled to the port while a reset message indicates no service is coupled to the port. The process then checks to see if the port address is the last possible port address on the computer (Block **218**). If it is not, the port address is incremented (Block **220**) and a new communication initiation message is sent to the next port address of the computer at the destination address (Block **206**). The process continues until all of the port addresses on a computer have been tested to determine whether a service is coupled to each port. After each port has been checked for a service, the process determines whether another destination address is available (Block **224**). If there is, another destination address is obtained (Block **200**) and the process continues at the first port address for the next computer. The process terminates when all of the computers on network **12** have been checked.

Another embodiment of the present invention is shown in FIG. **6**. In system **50**, a RPC message generator **52** and response evaluator **54** are coupled to transport layer **22**. RPC message generator **52** generates a data segment having a command for an RPC service which may not require an authorization check such as a password. Response message evaluator **54** determines from a message received in response to the RPC service command message whether an RPC service having little or no authorization check is available over the network. A record of this service may be provided to the system or network administrator.

The process implemented by system **50** is depicted in FIG. **7**. The process begins by obtaining a destination address for a computer on the network **12** from source/destination address generator **34** (Block **240**). The destination port address is initialized to the first port address on the computer at the destination address (Block **242**) and a first RPC service command is generated by RPC message generator **52** (Block **244**). Preferably, a CONNECT command which identifies the destination address and port address is issued to transport layer **22** (Block **248**). Once a communication connection has been established, transport layer **22** notifies RPC message generator **52** (Block **250**). RPC message generator **52** then passes the generated service command to transport layer **22** and a message containing the service command is transmitted to the port with which communication has been established (Block **252**). Response message evaluator **54** then waits for a response (Block **254**). If a response is detected which indicates the service command was executed (Block **258**), the destination address, port address and type of RPC service is stored in topology table **46** (Block **260**). If no communication connection was established with the port, no entry is made for the port. If communication is established but the port does not respond to the first service command, RPC message generator **52** determines if another RPC service command is available (Block **262**) and, if there is, it generates a service command for another service (Block **264**) and passes the command to transport layer **22** (Block **252**). There are a number of known RPC commands for the UNIX operating system and RPC

message generator **52** may generate a service command for each one to determine if it exists on a port being tested. If the process does not determine that an RPC service is coupled to the port, it identifies the service as a non-RPC service and stores an unknown or non-RPC service indicator in table **46** (Block **266**). Response evaluator **54** evaluates any message received which was responsive to the next service command (Blocks **254**, **258**). After the process finishes its interrogation of a port for the type of service coupled to the port, the process determines whether another port exists (Block **270**). If there are other ports to be interrogated, the port address is incremented (Block **272**) and the process continues until all the ports on the computer at the destination address have been tested. The process then continues by determining whether another destination address for a computer on the network exists (Block **276**) and, if it does, repeating the process for each port on that computer. When the process of FIG. **7** is completed, a topology map has been built which identifies the port and the RPC service coupled to each port for each computer on the network.

System **50** of FIG. **6** may be combined with system **40** of FIG. **4** such that once topology table **46** identifying those ports which are coupled to a service has been generated by response evaluator **44** of system **40**, RPC message generator **52** need only attempt to identify which of the ports identified as being coupled to a service are coupled to an RPC service having little or no authorization check. Response evaluator **54** of system **50** message generator may then identify the RPC services for those ports which respond to service commands generated by RPC message generator **52**.

An embodiment used to test the configuration of a router is shown in FIG. **8**. System **60** includes a communication message generator **62** and a response evaluator **64**. Preferably, communication message generator **62** includes a source routing verifier **66**, a source porting verifier **68** and a Socks configuration verifier **70**. Socks configuration verifier **70** and source routing verifier **66** execute in the application layer of a computer which is located outside network **12** and router RI which controls access to network **12**. Source porting verifier **68** specifies a source port for data messages being sent to a computer on network **12** and, consequently, it communicates with transport layer **22** and network layer **24** of protocol stack **20** on the computer executing the program which implements system **60**.

The process performed by the source routing verifier **66** is shown in FIG. **9**. That process begins by obtaining a destination address for a computer on network **12** from source/destination address generator **34** (Block **300**). The computer to which the message is to be ultimately delivered is defined by a destination address. The source address used to identify an intermediate source for a source routed message is also obtained from source/destination address generator **34** (Block **302**). Source routing verifier **66** then passes the source and destination addresses to transport layer **22** (Block **306**) to source route a message to a computer at the destination address on network **12** through the intermediate source identified by the source address (Block **310**). If a response is detected by response message evaluator **64** to the source routed message (Block **312**), a log indicating that the source routing blocking feature is not activated for the particular source/destination address combination is recorded in table **46** (Block **314**). If another source address is available for another computer on the network (Block **316**), it is obtained and another source routed message through the selected source address to the destination address is attempted. After attempts to source route messages to the destination address through all the source addresses for the other computers on the network have been attempted, the process determines if all destination addresses have been tested (Block **318**). If another destination address is available, another destination address is obtained and the process is repeated using the addresses of the other computers on the network as source addresses for source routed messages to the next destination address. In this manner, a log of all the source routed combinations which are not being blocked by the router are recorded in table **46** so the router may be reconfigured.

FIG. **10** shows a process implemented by source porting verifier **68**. The process begins by obtaining a destination address for a computer on the network from source/destination address generator **76** (Block **340**). Preferably, a source port address which corresponds to the default FTP source port address, typically port address **20**, is provided to network layer **24** (Block **342**). Until it is changed, data messages from the computer executing the program which implements the process of FIG. **11** generates data messages having a source port address of **20**. The destination port address is set to the first port address (Block **344**) and a data message having a source port address of **20** is sent to the port of the computer at the destination address (BLOCK **348**). Response evaluator **72** evaluates the responsive message received (Block **350**), if any, to determine whether the port responded to the source ported data message. Each response is stored in table **46** (Block **354**). The process determines if there is another destination port address (Block **358**) and, if there is, the destination port address is incremented (Block **360**). The process continues by checking the next destination port. If all the destination ports on the destination computer have been checked, the process determines if another source port address is to be tested (Block **364**). If there is, the next source port address is obtained (Block **366**) and the ports of the destination computer are tested with messages having the new source port address. Alternatively, all source port addresses may be exhaustively tested. If there are no more source port addresses to check, the process determines if another destination address exists on the network (Block **368**). If it does, the next destination address is obtained (Block **340**) and the process continues. Otherwise, the process stops.

A router may be configured with a rule which blocks data messages from computers external to network **12**. However, another rule may permit messages with certain source port address values to pass through in order to support certain services such as FTP. FTP requires a source port address of **20**. A hacker may attempt to get into a network by sending messages with a source port value which a router passes because it conforms to the rule for FTP messages. The process of FIG. **10** determines whether messages with predetermined source port addresses from computers external to the network are able to be received by computers on a network despite router configuration rules which would otherwise prevent the transmission of the messages.

As discussed above, Socks servers do not pass simply pass messages between computers on the network and those external to the network but instead require two separate communication connections. One communication connection is with an external computer and the other communication connection is with a computer on the network. In this manner, the Socks server may more thoroughly examine message in accordance with the rules configured for the server before passing the messages from one communication connection to another communication connection.

A preferred process implemented by the Socks configuration verifier of FIG. **8** is shown in FIG. **11**. That process

begins by having the computer executing the program which implements the process of FIG. **11** connect to the Socks server (Block **400**). A destination address is then obtained from the source/destination address generator **34** and used to request that the Socks server connect to the computer on the network at the destination address (Block **402**). The destination port address is set to the first port address value of the possible range of port address values (Block **406**). A service command is then generated (Block **410**) and a service command message addressed for the computer at the destination address is sent to the Socks server (Block **412**). The process then waits for a response (Block **416**). The response message is evaluated by response message generator **64** to determine if the response message indicates that the computer at the destination address received the service command (Block **420**). If it did not, the process determines if another communication method is available (Block **424**). If there is, the service command message is modified for another communication method (Block **426**) and sent to the Socks server (Block **412**). For example, if the message did not go through the Socks server, the service command message may be reformatted as a source routed message or a message with a predetermined source port value to see if the Socks server passes that type of message to the computer at the destination address. If no other communication format is available, the process continues by determining if another port address is available (Block **438**).

If the message indicates that the computer on the network responded to the service command, the process determines whether the service command was executed (Block **430**). If it was, the service and port address are stored in table **46** (Block **432**). If the response message indicates that the service command was received but not executed, the process determines if another service command is available (Block **434**). If there is, a new service command is generated (Block **410**) and the process continues until all service commands have been attempted for the port address at the destination address computer. If no other service commands remain to be tried, an indicator is stored in table **46** which indicates communication was established with the port address but no service was executed (Block **432**).

The process continues by determining if another port address remains for the computer at the destination address (Block **438**). If one does, the port address is incremented (Block **440**) and the testing for the new port address continues (Block **410**). Otherwise, the process determines whether another destination address is available on the network (Block **444**). If there is, it is obtained from source/destination address generator **34** (Block **402**) and testing of the computer at the new destination address continues. Otherwise, the communication connection with the Socks server is terminated and the process stops.

A more preferred embodiment of the present invention is shown in FIG. **12**. System **80** includes IP spoofing attack generator **32**, communication initiation message generator **42**, RPC message generator **52**, communication message generator **62**, source/destination address generator **34**, topology table or log **46** and protocol stack **20** which operate in manner consistent with the description of the embodiments for those like numbered components discussed above. System **80** also includes response evaluator **82** which includes the functionality of response message evaluators **44**, **54** and **64** as discussed above. A Graphic User Interface (GUI) **84** is also provided to accept input and control from a user and to display options and information to a user in a known manner. A user may use GUI **84** to activate each of the network verifiers **32**, **42**, **52** or **62** individually or selectively identify a group of verifiers to automatically execute and build the information in table **46**. GUI **84** also permits a user to enter information for execution of the verifiers such as defining or adding predetermined source port addresses, RPC services, addresses for computers added or deleted from a network or the like.

In operation, a user activates the program which implements an embodiment of the present invention such as system **80**. As a result, GUI **84** may present options to the user such as modifying information for system operation, selection of one or more of the network verifiers or display of stored information. After the user makes a selection, system **80** then performs the requested option. For example, if the user selects the system information modification option, the user is permitted to change system information such as adding addresses for new computers on a network. GUI **84** then returns the user to the main option menu following completion of the input of data and the user may now select one or more network verifiers to run. GUI **84** then selectively activates the selected network verifiers which communicate with protocol stack **20** to communicate messages between the computer executing system **80** and a computer on the network being tested or a router or a Socks server coupled to the network. When the verification tests or scans are completed, the user may select the display option and either view or print the information The user may then use the displayed information to add authorization checks to services or new rules to a Socks server or router.

While the present invention has been illustrated by the description of a number of embodiments and while the embodiments have been described in considerable detail, it is not the intention of the applicant to restrict or any way limit the scope of the appended claims to such detail. Additional advantages and modifications will readily appear to those skilled in the art. The invention in its broader aspects is therefore not limited to the specific details, representative systems and methods, and illustrative examples shown and described. Accordingly, departures may be made from such details without departing from the spirit or scope of applicant's general inventive concept.

What is claimed is:

1. A system for detecting a security vulnerability in open network communications comprising:

an internet protocol (IP) spoofing attack generator for generating an IP spoofing attack on a target computer coupled to an open network to determine whether said target computer is vulnerable to an IP spoofing attack which emulates communication from another computer on said open network;

a service command message generator for generating a service command to be executed by a service coupled to a port on said target computer; and

said IP spoofing attack generator transmitting said service command to said target computer to generate a response in said target computer that provides a compromise indication without altering system operational parameters of said target computer.

2. The system of claim **1**, wherein said generated service command is for one of an rsh and an rlogin service to determine whether authorization checks for said service exist.

3. The system of claim **2**, wherein said generated service command causes said target computer to generate an electronic mail message indicative that said target computer has been compromised.

4. The system of claim **3**, wherein said generated service command causes said target computer to initiate a Telnet

session with a computer which logs said Telnet session to indicate said target computer has been compromised.

5. The system of claim 1, further comprising:

a source/destination address generator which generates source and destination addresses for messages corresponding to an open network protocol used to communicate on said open network, said destination address corresponding to said target computer and said source address corresponding to said computer being emulated for said attack.

6. The system of claim 5, wherein said source/destination address generator generates source and destination address combinations which are used by said IP spoofing attack generator to test vulnerability of each computer in said open network to an IP spoofing attack which emulates communication from each of said other computers on said open network.

7. A system for generating a service topology map for each computer on an open network without completing a communication connection with any computer on the open network comprising:

a communication initiation message generator for generating communication initiation messages, said communication initiation messages being transmitted to ports on a computer on an open network; and

a response message evaluator for determining from response messages received from said ports receiving said communication initiation messages whether services exist on said ports receiving said communication initiation messages, said response messages not completing communication connections with said ports so that services coupled to said ports may be detected without completing communication connection with said ports.

8. The system of claim 7, further comprising:

a table for storing service indicators indicative of which ports responding to said communication initiation messages are coupled to services.

9. The system of claim 8, wherein said communication initiation message generator generates a communication initiation message for each port address on a computer on said open network.

10. The system of claim 9, wherein a source/destination address generator generates a destination address for each computer on an open network so that each port on each computer on said open network receives a communication initiation message and said table contains service indicators for each port of each computer on said open network which responds to said communication initiation messages.

11. The system of claim 7, wherein said communication initiation message generator generates sync messages for a TCP/IP protocol.

12. The system of claim 11, wherein said response message evaluator determines a service is coupled to a port receiving a communication initiation message in response to detecting a sync/ack message.

13. The system of claim 7, wherein said communication initiation message is the first message for a three handshake protocol to establish a communication connection.

14. A system for detecting vulnerability of ports coupled to remote procedure call (RPC) services on a computer of an open network comprising:

a remote procedure call (RPC) message generator for generating and sending RPC service commands to ports on a computer on an open network; and

a response message evaluator for evaluating response messages from said ports of said computer receiving said RPC service commands, said response messages indicating whether said RPC service commands were executed by an RPC service coupled to said ports of said computer receiving said RPC service commands without establishing a communication connection with said ports.

15. The system of claim 14, further comprising:

a table for storing port addresses and service indicators that indicate which particular RPC services are coupled to ports receiving said service commands.

16. A system for detecting vulnerabilities in routers comprising:

a communication message generator for generating and sending service commands from a computer external to an open network to ports on computers coupled to said open network through a router; and

a response message evaluator for evaluating response messages received from said ports on computers of said open network in response to said service commands sent from said communication message generator external to said open network whereby access to said computers on said open network through said router may be determined without referencing configuration files of said router.

17. The system of claim 16, wherein said communication message generator includes a source routing verifier for generating source routed messages with a destination address of a computer on said open network and an intermediate source address on said open network; and

said response message evaluator evaluating response messages received from said ports on computers of said open network in response to said service commands sent from said communication message generator external to said open network to detect a vulnerability in said router of permitting source routed messages to bypass rules configured for filtering inbound messages on said router.

18. The system of claim 17, wherein each source address for each computer on said open network is used as said intermediate source address with each destination address for each computer on said open network to test each possible intermediate source/destination address combination for source routed messages on said open network.

19. The system of claim 18, further comprising:

a table for storing indicators for each intermediate source address/destination address combination that is detected as being vulnerable to receiving source routed messages.

20. The system of claim 16, wherein said communication message generator includes a source porting verifier for generating service command messages with a source port address having a predetermined value; and

said response message evaluator evaluating response messages received from said ports on computers of said open network in response to said service command messages having said predetermined source port address values sent from said source porting verifier external to said open network to detect said router passing messages having said predetermined source port address values to ports coupled to services on said open network.

21. The system of claim 20, wherein service command messages having said predetermined source port address value are sent to each computer on said open network.

22. The system of claim 21, further comprising:

a table for storing service indicators for each computer address that is detected as being vulnerable to receiving source ported messages.

23. The system of claim 22, wherein said predetermined value corresponds to a default source port address for a file transfer protocol (FTP) message of a TCP/IP protocol.

24. The system of claim 16, further comprising:

a Socks configuration verifier for establishing a communication connection with a Socks server and for sending service command messages to computers on said open network coupled to said Socks server; and

said response message evaluator evaluating said messages received in response to said service command messages to determine whether said service command message was passed by said Socks server to one of said computers on said open network.

25. The system of claim 24 said response message evaluator determining whether said service command message was executed by said one computer on said open network.

26. The system of claim 25 said response message evaluator storing service indicators indicative of said services which executed said service command messages received at said port addresses.

27. A method for detecting a security vulnerability in an open network comprised of the steps of:

attempting an Internet Protocol (IP) spoofing attack against a target computer and open network;

generating a service command message; and

sending said service command message to said target computer following said IP spoofing attack to determine whether said target computer has been compromised, said service command message generating an indicator of the success of the IP spoofing attack without altering the operational parameters of the target computer.

28. The method of claim 27, wherein said generating service command message step generates one of an rsh and rlogin command.

29. The method of claim 28, wherein said generating step:

generates an electronic mail message indicative of the success of the IP spoofing attack in response to said service command message.

30. The method of claim 27, further comprising the step of:

initiating a Telnet session between said target computer and another computer to indicate the success of said IP spoofing attack in response to said service command message.

31. The method of claim 27, further comprising the steps of:

generating source addresses and destination addresses for said IP spoofing attack; and

attempting said IP spoofing attack against each said generated destination address by emulating communication from each of said source addresses.

32. A method for generating a service topology map of an open network comprising the steps of:

generating a communication command initiation message;

sending said communication command initiation message to a port on a computer on an open network;

receiving a message from said port in response to said communication initiation message being received at said port; and

evaluating said message received from said port to determine whether a service is coupled to said port without establishing a communication connection with said port.

33. The method of claim 32, further comprising the step of:

storing a service indicator to provide a reference that said port has a service coupled thereto which may be accessed from another computer.

34. A method for detecting availability of a service on a port of a computer on an open network comprising the steps of:

generating a service command message;

sending said generated service command message to a port of a computer on said open network;

receiving a message from said port in response to said port receiving said generated service command message; and

evaluating said message received from said port to determine whether a service coupled to said port executed said service command message, without establishing a communication connection with said ports.

35. The method of claim 34, further comprising the step of:

storing a service indicator indicative that said service coupled to said port executed said service command message.

36. The method of claim 35, wherein said generating step generates service command messages for different services; and

said evaluating step determines the type of service coupled to said port which executed said service command message.

37. The method of claim 36, wherein said generating step generates said service command messages for each port of a computer of said open network.

38. The method of claim 34, further comprising the steps of:

establishing a communication connection with a Socks server;

requesting said Socks server establish a communication connection with a computer on said open network; and

said evaluating step determining whether said Socks server is configured to stop said service command message from being sent to said port of said computer of said open network.

39. The method of claim 34, wherein said generating step generates remote procedure call (RPC) service command messages.

40. The method of claim 34, wherein said generating step generates service command messages having predetermined source port addresses.

41. The method of claim 34, wherein said generating step generates source routed service command messages.

* * * * *

Exhibit A-4

(12) **United States Patent**
Shostack et al.

(10) **Patent No.: US 6,298,445 B1**
(45) **Date of Patent: Oct. 2, 2001**

(54) **COMPUTER SECURITY**

(75) Inventors: **Adam Shostack**, Boston, MA (US); **David Allouch**, Givat Shimuel (IL)

(73) Assignee: **Netect, Ltd.** (IL)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/070,698**

(22) Filed: **Apr. 30, 1998**

(51) **Int. Cl.**[7] **G06F 12/14**
(52) **U.S. Cl.** **713/201**; 713/200
(58) **Field of Search** 380/21; 713/200, 713/201, 202; 340/825.31, 825.34; 709/229, 235

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,405,829		9/1983	Rivest et al.	178/22.1
5,361,359		11/1994	Tajalli et al.	395/700
5,557,346		9/1996	Lipner et al.	380/21
5,557,742		9/1996	Smaha et al.	395/186
5,557,765		9/1996	Lipner et al.	380/21
5,640,454		6/1997	Lipner et al.	380/21
5,684,957		11/1997	Kondo et al.	395/200.06
5,825,891	*	10/1998	Levesque et al.	380/49
5,867,647	*	2/1999	Haigh et al.	713/200
5,898,784	*	4/1999	Kirby et al.	380/49
5,987,611	*	11/1999	Freund	713/201
6,003,084	*	12/1999	Green et al.	709/227
6,061,796	*	5/2000	Chen et al.	713/201
6,219,652	*	4/2001	Carter et al.	713/200

FOREIGN PATENT DOCUMENTS

0329415A2 2/1989 (EP) G06F/1/00

OTHER PUBLICATIONS

PR Newswire, Minneapolis, MN (Sep. 1995) "Network Systems and Haystack Labs Introduce Netstalker to Track Hacker Attempts" Accession No. 00531881 (3 pgs.).

PR Newswire, Atlanta, GA (Sep. 1995) "HP Selects NSC Borderguard Firewall Solution for Web Servers" Accession No. 00531230 (2 pgs.).

Business Wire, Lexington, MA (Oct. 1995) "Internet Security Signs Master Distributor Agreement with Bellcore, Will license Bellcore's SysGuard (TM) and Pingware (TM) Software Tools" Accession No. 00542376 (2 pgs.).

Business Wire, Las Vegas, NV (Apr. 1996) "WheelGroup to Demonstrate Real–Time Remote Network Monitoring Via Secure Virtual Private Networks on the Internet, Network Systems Corp. and Haystack Labs Products Combine to Create Secure Virtual Private Nets" Accession No. 00597652 (2 pgs.).

News Release, Vienna, VA (May 1996) "BTG Gains Exclusive Right to Offer Electronic Data Security System to Internet and Network Users in the Federal Market" Accession No. 00634843 (2 pgs.).

Business Wire, Vienna, VA (May 1996) "BTG Invests in WheelGroup's Information Protection System" Accession No. 00612063 (2 pgs.).

Business Wire, Alameda, CA (Jun. 1996) "New Firewall Technology from Ascend Substantially Lowers the Price of Remote Network Security; New Industry Price Point Makes Enterprise—Wide Security Viable" Accession No. 00627510 (3 pgs.).

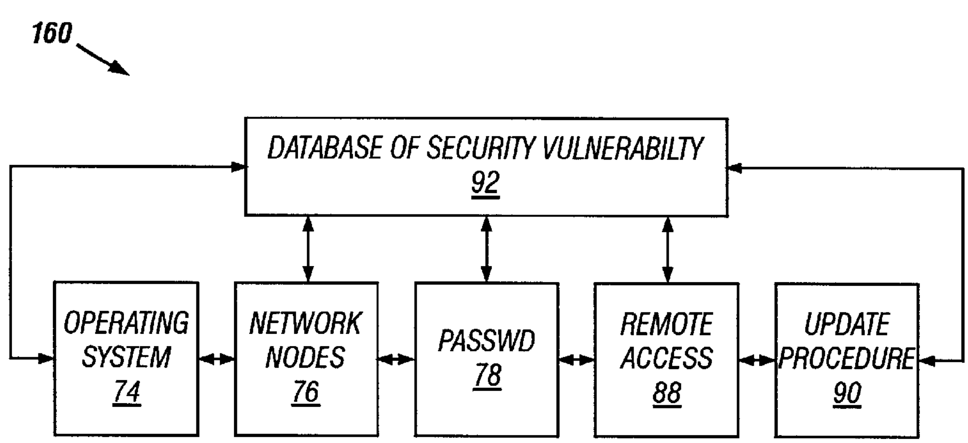
(List continued on next page.)

Primary Examiner—James P. Trammell
Assistant Examiner—Pierre E. Elisca
(74) *Attorney, Agent, or Firm*—Hugh R. Kress; Winstead Sechrest & Minick

(57) **ABSTRACT**

In one aspect, the invention relates to automatically providing enhancements to computer security software whenever the enhancement becomes available. In another aspect, the invention relates to an integrated system for assessing security vulnerabilities of a computer and/or a computer network.

14 Claims, 5 Drawing Sheets



OTHER PUBLICATIONS

Business Wire, Austin, TX (Jun. 1996) "Venrock Associates and Trellis Management Co. Invest in Haystack Labs' Website Security Technology" Accession No. 00624135 (2 pgs.).
Business Wire, Glenwood, MD (Oct. 1996) "Trusted Information Systems' Gauntlet Internet Firewall Passes NSA Test" Accession No. 00679353 (2 pgs.).
Business Wire, Alameda, CA (Oct. 1996) "Ascend's Secure Access Firewall Solution Receives NCSA Certification, Ascend's Intergrated Firewall Option for MAX and Pipeline Products Meets Rigorous NCSA Standards" Accession No. 00680964 (2 pgs.).
Business Wire, Austin, TX (Oct. 1996) "Haystack Labs Licenses Its Active Security Technology to Sun, Haystack Labs Awarded US Patent for its Intrusion Detection Software" Accession No. 00679097 (3 pgs.).
Business Wire, San Antonio (Nov. 1996) "WheelGroup and BTG Announce First Large–Scale Deployment of NetRanger™ Network Security Management System" Accession No. 00695074 (2 pgs.).
Business Wire, Austin, TX (Nov. 1996) "Haystack Labs Ships WebStalker–Pro 1.0" Accession No. 00695472 (3 pgs.).
Business Wire, Austin, TX (Dec. 1996) "Haystack Labs launches its Partner Program and Garners 10 Charter Members" Accession No. 00706659 (3pgs.).
Business Wire, New York, NY (Dec. 1996) "Haystack Labs unveils WebStalker–Pro NT, 'Beyond–the–Firewall' Security Product for Microsoft Windows NT" Accession No. 00707375 (3 pgs.).
Business Wire, Austin, TX (Dec. 1996) "Haystack Labs to Debut WebStalker for Windows NT and Present the "Top Ten Reasons Why a Firewall is Not Enough"" Accession No. 00705316 (2 pgs.).
PR Newswire, Comnet, WA (Feb. 1997) "Check Point Software and Haystack Labs Partner to Provide Advanced Suspicious Activity Monitoring Capabilities" Accesssion No. 00732995 (3 pgs.).
Business Wire, New York, NY (Mar. 1997) "Haystack Labs Appoints Former Security Dynamics Executive James M. Geary as President, Alisa Nessler as Vice President of Marketing" Accession No. 00748101 (3 pgs.).
Business Wire, Minneapolis, MN (May 1997) "StorageTek Network Systems Group, WheelGroup Thwart Hackers" Accession No. 00774595 (3 pgs.).
Business Wire, San Antonio, TX (May 1997) "WheelGroup Announces NetRanger for BorderGuard for Network Security Beyond Firewalls" Accession No. 00778276 (3 pgs.).
Business Wire, New York, NY (May 1997) Ernst & Young, LLP and WheelGroup Corp. Form Alliance to Provide Network Security Solutions. Accession No. 00784112 (2 pgs.).
Business Wire, El Segundo, CA (Jun. 1997) "Merisel Open Computing Alliance and Haystack Labs Inc. Offer Active Security–Product Suite, Provide New Level of Protection for Open Computer Systems" Accession No. 00788181 (2 pgs.).
Business Wire, Austin, TX (Jun. 1997) "Haystack Labs Announces Agreement with IBM to Resell WebStalker–Pro" Accession No. 00790002 (3 pgs.).
Business Wire, San Jose, CA (Jul. 1997) "Navigist Announces Network Security Review Service" Accession No. 00797507 (2 pgs.).
Business Wire, San Antonio, TX (Aug. 1997) "WheelGroup Releases NetRanger Version 2.0. Bringing Intrusion Detection to the Mainstream" Accession No. 00811341 (2 pgs.).
Ken Phillips, "Netective Nixes Ne'er–Do–Wells" Reprint from Aug. 4, 1997, PCWeek (4 pgs.).
NETECT "Company Profile" Tradeshow Pamphlet May 5–9, 1997 (7 pgs.).
Business Wire, San Antonio, TX (Oct. 1997) "WheelGroup Announces NetSonar Vulnerability Scanner, Best–of–Breed Consulting Expertise at Core of Product" Accession No. 00828916 (3 pgs.).
PR Newswire, Tampa, FL (Oct. 1997) "The Buy, Sell or Hold Company Starts Accent Software International with a Buy Rating" Accession No. 00824720 (4 pgs.).
PR Newswire, Glenwood, MD (Oct. 1997) "Trusted Information Systems, Inc. to Acquire Haystack Laboratories, Inc." Accession No. 00820649 (3 pgs.).
Pr Newswire, Cedar Grove, NJ (Dec. 1997) "Right to Privacy for Sale in Cyberspace, SynData Technologies Inc. Speaks Out Against Key Recovery" Accession No. 00835722 (2 pgs.).
Business Wire, Santa Clara, CA (Dec. 1997) "Network Associates Offers Web–Based Training on Sniffer Basics, Anytime, Anywhere Learning Through a Browser." Accession No. 00837415 (2 pgs.).
NETECT Inc. "Network Associates Teams with NETECT to Deliver Security Updates via the Internet" Press Release San Francisco, CA Jan. 13, 1998 (2 pgs.).
Business Wire, San Mateo, CA (Jan. 1998) "Keynote Systems to Add Remote Internet Performance Measurements to Network Associates WebSniffer: First OEM Relationship for Keynote Systems Perspective" Accession No. 00848145 (3 pgs.).
PR Newswire, Santa Clara, CA (Jan. 1998) "Network Associates Releases New Version of PC Medic—The First PC Diagnostic, Repair Product Integrated with Help Desk Technology" Accession No. 00845704 (3 pgs.).
PR Newswire, Washington, D.C. (Jan. 1998) "Network Associates Announces Industry's First and Only Distributed, Enterprise ATM Management System" Accession No. 00848419 (3 pgs.).
Business Wire, San Jose, CA (Feb. 1998) "Cisco Systems to Acquire WheelGroup Corporation, Cisco Extends Leadership in End–To–End Network Security Products." Accession No. 00857535 (2 pgs.).
PR Newswire, New York, NY (Feb. 1998) "Think Named Global Marketing and Communications Company of Record for Network Associates" Accession No. 00857760 (3 pgs.).
WheelGroup Corporation; Web Site (www.wheelgroup.com), 2 pages printed on Mar. 12, 1998.
WheelGroup Corporation, Web Site (www.wheelgroup.com/about/about.html), 8 pages printed on Mar. 12, 1998.
PR Newswire, Santa Clara, CA (Mar. 1998) "Network Associates Nuts & Bolts Wins PC Magazines Editor's Choice Award for Best Troubleshooting Utility" Accession No. 00863088 (2 pgs.).
PR Newswire, Santa Clara, CA (Mar. 1998) "Network Associates Launches Nuts & Bolts Deluxe with One of the Industry's First Year 2000 Hardware Fix Capabilities" Accession No. 00863089 (3 pgs.).
PR Newswire, Holmdel, NJ (Mar. 1998) "Updata Capital Co–Manages Merger of Trusted Information Systems (TIS) and Network Associates, Inc." Accession No. 00863686 (2 pgs.).

Common Gateway Interface, Web Site (http://hoohoo.ncsa.uiuc.edu/cgi/intro.html) 2 pages printed on Mar. 10, 1998.
Attack Pathology, Web Site (http://www.geek–girl.com/ids/0443.html), 3 pages printed on Mar. 12, 1998.
What SATAN is, Web Site (www.cs.ruu.nl/cert–uu/satan.html) 3 pages printed on Mar. 12, 1998.
Pingware info—was "Re: newbie intro", Web Site (http://www.geek–girl.com/ids/0325.html), 1 page printed on Mar. 12, 1998.
Security Administrator's Tool for Analyzing Networks, Web Site (www.fish.com/satan/) 3 pages printed on Mar. 12, 1998 with a copyright date of 1995.
About Axent, Web Site (http://www.axent.com/about/about.htm) 1 page printed on Mar. 12, 1998 with a copyright date of 1997.
Axent Company Overview. Web Site (www.axent.com) 2 pages printed on Mar. 12, 1998.
Axent Corporate Overview, Web Site (www.axent.com) 1 page printed on Mar. 12, 1998.
OmniGuard/ESM, Web Site (www.axent.com/product/esm/esm.htm) 10 pages printed on Mar. 12, 1998.
OmniGuard/ESM Brochure, Web Site (www.axent.com/product/esm/esm1.htm) 5 pages printed on Mar. 12, 1998.
OmniGuard/Intruder Alert, Web Site (www.axent.com/product/ita/its.htm) 10 pages printed on Mar. 12, 1998.
OmniGuard/URM UNIX Resource Manager, Web Site (www.axent.com/product/eacu/eacu.htm) 2 pages printed on Mar. 12, 1998.
OmniGuard/URM Brochure Enterprise Resource Manager, Web Site (www.axent.com/product/erm/erm2.htm) 5 pages printed on Mar. 12, 1998.
OmniGuard/URM Brochure UNIX Resource Manager, Web Site (www.axent.com/product/eacu/eacu1.htm) 4 pages printed on Mar. 12, 1998.
OmniGuard/PCShield, Web Site (www.axent.com/product/pcs/pcs.htm) 3 pages printed on Mar. 12, 1998.
OmniGuard/PCShield Brochure, Web Site (www.axent-.com/product/pcs/pcs1.htm) 4 pages printed on Mar. 12, 1998.
Haystack Labs TIS, Web Page (www.haystack.com) 1 page printed on Mar. 12, 1998.
OmniGuard UPM UNIX Privilege Manager, Web Site (www.axent.com/product/upm/upm.htm) 2 pages printed on Mar. 12, 1998.
OmniGuard/UPM Brochure UNIX Privilege Manager, Web Site (www.axent.com/product/upm/upm2.htm) 3 pages printed on Mar. 12, 1998.
OminGuard/Defender Secure Authentication Solution, Web Site (http://www.axent.com/product/def.htm) 1 page printed on Mar. 12, 1998; copyright 1997.
OmniGuard/Defender Brochure Secure Authentication Solution, Web Site (www.axent.com/product/def2.htm) 5 pages printed on Mar. 12, 1998.
OmniGuard/Power VPN Power Virtual Private Network, Web Site (http://www.axent.com/product/vpn/vpn.htm) 1 page printed on Mar. 12, 1998 with a copyright date of 1997.
OmniGuard/Power VPN Brochure Power Virtual Private Network, Web Site (http://www.axent.com/product/vpn/vpn1.htm) 5 pages printed on Mar. 12, 1998 with a copyright date of 1997.
System Security Scanner Datasheet, Web Site (http://iss.net/prod/s3ds.html) 4 pages printed on Mar. 12, 1998; copyright 1997.
RealSecure Datasheet, Web Site (http://iss.net/prod/rsds.html) 4 pages printed on Mar. 12, 1998; copyright 1997.
Internet Scanner Datasheet, Web Site (http://iss.net/prod/isds.html) 4 pages printed on Mar. 12, 1998; copyright 1997.
ISS Product Literature, Web Site (http://iss.net/prod/datashts.html) 1 page printed on Mar. 12, 1998; copyright 1994–1998.
ISS in the News, Web Site (http://www.intsysserv.com/body__default.htm) 1 page printed on Mar. 12, 1998.
Welcome to ISS Online, Web Site (http://www.isscorp.com) 1 page printed on Mar. 12, 1998; copyright 1998.
ISS Products: Software Solutions, Web Site (http://www.isscorp.com/products.html) 2 pages printed on Mar. 12, 1998; copyright 1998.
About Haystack Labs, Web Site (http://www.haystack.com/about.htm) 1 page printed on Mar. 12, 1998.
Haystack Labs Company Background, Web Site (http://www.haystack.com/background.htm) 1 page printed on Mar. 12, 1998.
Haystack Laboratories Copyright and Trademarks, Web Site (http://www.haystack.com/trademar.htm) 1 page printed on Mar. 12, 1998.
Haystack Laboratories Company Timeline, Web Site (http://www.haystack.com/timeline4.htm) 1 page printed on Mar. 12, 1998.
Haystack Labs Products, Web Site (http://www.haystack-.com/products.htm) 1 page printed on Mar. 12, 1998.
WebStalker, Web Site (http://www.haystack.com/webstalk.htm) 2 pages printed on Mar. 29, 1998.
WebStalker–Pro Tour, Web Site (http://www.haystack.com/wstour.htm) 2 pages printed on Mar. 29, 1998.
WebStalker–Pro, Web Site (http://www.haystack.com/ws__demo/htm/home.htm) 1 page printed on Mar. 12, 1998 with a copyright date of 1993–96.
Sample the WebStalker Pro Interview, Web Site (http://www.haystack.com/sample.htm) 1 page printed on Mar. 12, 1998.
Stalker Product, Web Site (http://www.haystack.com/stalk.htm) 1 page printed on Mar. 12, 1998.
Stalker Key Features, Web Site (http://www.haystack.com/s–feature.htm) 2 pages printed on Mar. 12, 1998.
Stalker Product Details, Web Site (http://www.haystack-.com/s–detail.htm) 1 page printed on Mar. 12, 1998.
Stalker Tracer/Browser, Web Site (http://www.haystack-.com/tracer.htm) 2 pages printed on Mar. 29, 1998.
Stalker Misuse Detector, Web Site (http://www.haystack-.com/misuse.htm) 1 page printed on Mar. 12, 1998.
Stalker Audit Control, Web Site (http://www.haystack.com/s–audit.htm) 1 page printed on Mar. 12, 1998.
Stalker Storage Manager, Web Site (http://www.haystack-.com/s–manage.htm) 1 page printed on Mar. 12, 1998.
What's New at Haystack Labs, Web Site (http://www.haystack.com/watnu.htm) 2 pages printed on Mar. 12, 1998.
Navigational Bar for Haystack Labs, Web Site (http://www.haystack.com/map.htm) 1 page printed on Mar. 12, 1998.
Security Issues, Web Site (http://www.haystack.com/satan.htm) 3 pages printed on Mar. 12, 1998.
Distributors, Web Site (http://www.haystack.com/distrib.htm) 1 page printed on Mar. 12, 1998.
Network Associates: Main Index, Web Site (http://www.networkassociates.com/main.htm?) 1 page printed on Mar. 12, 1998 with a copyright date of 1997.

Network Associates: Technical Links, Web Site (http://www.netassociates.com/Resources/TechLinks.htm) 2 pages printed on Mar. 12, 1998; copyryright 1997.
Micah Development's—Full Armor Access Control Software, Web Site (http://www.fullarmor.com/) 1 page printed on Mar. 12, 1998.
Full Armor Prducts by Micah, Web Site (http://www.fullarmor.com/products/index.html) 3 pages printed on Mar. 12, 1998; copyright 1996, 1997 Micah Development Corporation.
Network Associates: Desktop Management, Web Site (http://www.netassociates.com/Resources/Software/SoftDesk.htm) 1 page printed on Mar. 12, 1998; copyright 1997.
Network Associates: Virus Detectors Web Site (http://www.netassociates.com/Resources/Software/SoftVirus.htm) 1 page printed on Mar. 12, 1998; copyright 1997.
Network Associates Valuable Free Software & Demos, Web Site (http://www.netassociates.com/Resources/Software/SoftValuable.htm) 1 page printed on Mar. 12, 1998; copyright 1997.
Network Associates: The Networking Company, Web Site (www.netassociates.com) 1 page printed on Mar. 30, 1998.
Products, Web Site (http://www.axent.com/product/products.htm) 1 page printed on Mar. 16, 1998; copyright 1997–98.
OmniGuard/Intruder Alert, Web Site (http://www.axent.com/product/ita/ita.htm) 9 pages printed on Mar. 16, 1998; copyright 1998.
How Secure is the Information on Your Computer System?, Web Site (http://www.axent.com/product/overview/corporate.htm) 12 pages printed on Mar. 16,1998; copyright 1997.
OmniGuard/ESM Enterprise Security Manager, Web Site (http://www.axent.com/product/esm/esm.htm) 10 pages printed on Mar. 16, 1998; copyright 1997.
OmniGuard/ESM Brochure Enterprise Security Manager, Web Site (http://www.axent.com/product/esm/esm1.htm) 5 pages printed on Mar. 16, 1998; copyright 1997.
OmniGuard/ESM Enterprise Security Manager Web Site, (http://www.axent.com/support/techsup/esmupd.htm) 4 pages printed on Mar. 16, 1998; copyright 1997.
OmniGuard/Intruder Alert Brochure, Web Site (http://www.axert.com/product/ita/ita1.htm) 4 pages printed on Mar. 16, 1998; copyright 1998.
OmniGuard/ERM Enterprise Resource Manager, Web Site (http://www.axent.com/product/erm/erm.htm) 1 page printed on Mar. 12, 1998; copyright 1997.
OmniGuard/NetReconWeb Site (http:www.axent.com/netrecon/) 3 pages printed on Mar. 16, 1998; copyright 1997–1998.
OmniGuard/NetRecon Brochure, Web Site (http://www.axent.com/netrecon_brochure.htm) 6 pages printed on Mar. 16, 1998; copyright 1997–1998.
Netect Products, Web Site (www.netect.com/product.htm) 5 pages printed on Mar. 30, 1998; copyright 1997.
Spangler, T. "Netective Emulates Network Attacks, Hacker Methods" (http://www.netect.com/artic104.htm) 2 pages printed on Apr. 29, 1998.
Network Associates Teams with NETECT to Delivery Security Updates Via the Internet, Web Site (http://netect.com/news06.htm) 2 pages printed on Mar. 30, 1998; copyright 1997.
NETECT Demonstrates that Encryption Alone Can Not Protect Against Cyberthreats, Web Site (http://www.netect.com/news05.htm) 2 pages printed on Mar. 30, 1998; copyright 1997.
NETECT Delivers First Single–Product Solution for Protecting Networks from Internal and External Cyberthreats, Web Site (http://www.netect.com/news04.htm) 2 pages printed on Mar. 30, 1998; copyright 1997.
NETECT Delivers OPSEC—Enabled Security Technology to Work with Check Point Firewall–1, Web Site (http://www.netect.com/news01.htm) 2 pages printed on Mar. 30, 1998; copyright 1997.
Intruder–Attack Simulators, Web Site (http://www.netect.com/articl05.htm) 2 pages printed on Mar. 30, 1998; copyright 1997.
Netective's On the Case For Security, Web Site (http://www.netect.com/articl103.htm) 1 page printed on Mar. 30, 1998; copyright 1997.
Netect's Netective Simulates Hacker Attacks, Letting Corporate Networkers Shore Up Weak Spots, Web Site (http://www.netect.com/articl07.htm) 2 pages printed on Mar. 30, 1998; copyright 1998.
HTML Documents: A Mosaic Tutorial, Web Site (http://sti.larc.nasa.gov/demos/html__tutorial.html) 16 pages printed on Apr. 7, 1998; copyright 1997.
FTP Tutorial, Web Site (http://www.paccd.cc.ca.us/instadmn/physcidv/ftp__tutr.htm) 4 pages printed on Apr. 7, 1998.
Internet World Dec. 1997—Suite Decisions, Web Site (http://search.internet.com/plweb–c...netWorld+718+3+wAAA+push%26channel) 7 pages printed on Apr. 9, 1998; copyright 1997.
What is IMAP, Web Site (http://www.imap.org/whatisIMAP.html) 1 page printed on Apr. 9, 1998; copyright 1996–1998.
Detecting Net Trouble Spots, Web Site (http://www.netect.com/articl02.htm) 2 pages printed on Mar.30, 1998; copyright 1997.
Update: Push Channel Pays Dividend As Marketing Tool For Software Maker, Web Site (http://www.internetworld.com/print.../07/21/markcomm/19970721–push.html) 1 page printed on Apr. 9, 1998.
Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP, Web Site (http://www.imap.org/imap.vs.pop.brief.html) 4 pages printed on Apr. 9, 1998; copyright 1996.
Message Access Paradigms and Protocols, Web Site (http://www.imap.com/imap.vs.pop.html) 10 pages printed on Apr. 9, 1998; copyright 1996.
MD5 Checksum Utility, Web Site (http://www.threel.co.uk/tech/tools/md5.htm) 1 page printed on Apr. 10, 1998.
Frequently Asked Questions, Web Site (http://hoohoo.ncsa.uiuc.edu/docs/FAQ.html#whatis) 5 pages printed on Apr. 10, 1998.
How does PGP authentication Work?, Web Site (http://kekec.e5.ijs.si/security/efh/howauth.html) 1 page printed on Apr. 10, 1998.
RSA Algorithm Javascript Page, Web Site (wysiwyg://4/http://www.engr.orst.edu.~makmur/HCproject/) 2 pages printed on Apr. 11, 1998.
Digital Signatures, Web Site (http://csrc.ncsl.nistpubs/800–7/node211.html) 2 pages printed on Apr. 11, 1998.
Authentication: The Key to Internet Security, Web Site (http://www.cs.rpi.edu/~noel/Security/Authenticate.html) 2 pages printed on Apr. 11, 1998.

Question 143. What are Identification Schemes and Authentication Protocols?, Web Site (http://www.rsa.com/rsalabs/newfaq/q143.html) 1 page printed on Apr. 11, 1998; copyright 1996.

Question 3. What is Public–Key Cryptography? Web Site (http://www.rsa.com/rsalabs/newfaq/q3.html) 2 pages printed on Apr. 11, 1998; copyright 1996.

rfc 1321, Web Site (http://www.cis.ohio–state.edu/htbin/rfc/rfc1321) 18 pages printed on Apr. 12, 1998.

WindowsNT Bonk Update, Web Site (http://www.ndsu.nodak.edu/csg/info/bonknt.html) 2 pages printed on Apr. 14, 1998.

Craig, A., New Tools Make Hackers Lives Easy, Web Site (wysiwyg://60/http://www.crystosoft.com/snews/dec97/24129700.htm) 2 pages printed on Apr. 14, 1998; copyright 1995–1998.

Denial of Service or "Nuke" Attacks, Web Site (http://deckard.mc.duke.edu/irchelp/nuke/) 2 pages printed on Apr. 14, 1998; copyright 1996–1997.

User Datagram Protocol, Web Site (http://www.alexia.net.au/~www/yendor/internetinfo/udp.html) 3 pages printed on Apr. 14, 1998.

Syn Flood, Web Site (http://www.axent.com/swat/1attacks/unix/generic/synflood.htm) 1 page printed on Apr. 14, 1998.

Phrack Magazine, Web Site (http://www.fc.net/phrack/files/p48/p48–13.html) 20 pages printed on Apr. 14, 1998.

Outsmarting Intruders, Web Site (http://www.netect.com/plug.htm) 1 page printed on Apr. 15, 1998; copyright 1997.

Channel Definition Format (CDF), Web Site (http://www.w3.org/TR/NOTE–CDFsubunit.html#Introduction) 12 pages printed on Apr. 17, 1998; copyright 1997.

Electronic Mail re: PRNewswire "Network Guardians Introduces Java–based Network Security Scanner" 2 pages; copyright 1998.

Electronic Mail re: The Digital Commerce Society of Boston Presents "No Silver Bullet" Digital Commerce and Payment Security, Apr. 7, 1997 at the Downtown Harvard Club of Boston (4 pgs.).

Electronic Mail re: "Everything You Ever Wanted to Know About Security Holes, for a Price" printed on Apr. 13, 1998 (1 pg.).

* cited by examiner

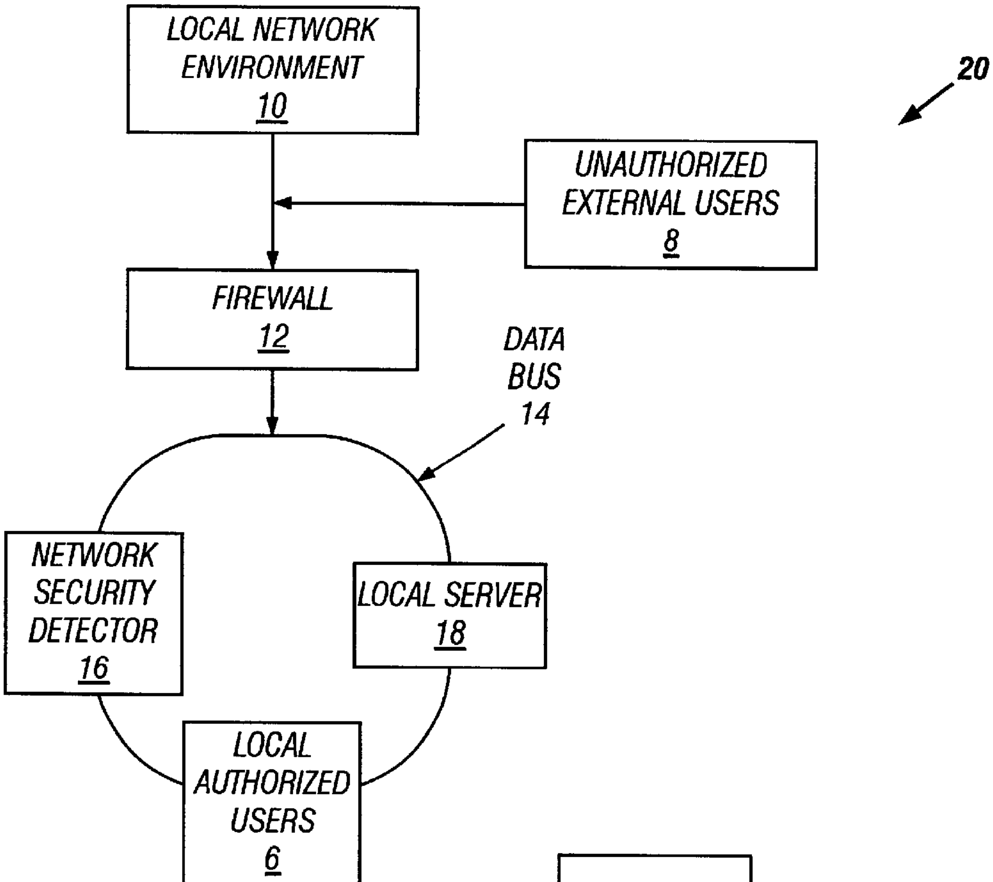

20
LOCAL NETWORK ENVIRONMENT 10
UNAUTHORIZED EXTERNAL USERS 8
FIREWALL 12
DATA BUS 14
NETWORK SECURITY DETECTOR 16
LOCAL SERVER 18
LOCAL AUTHORIZED USERS 6

FIG. 1

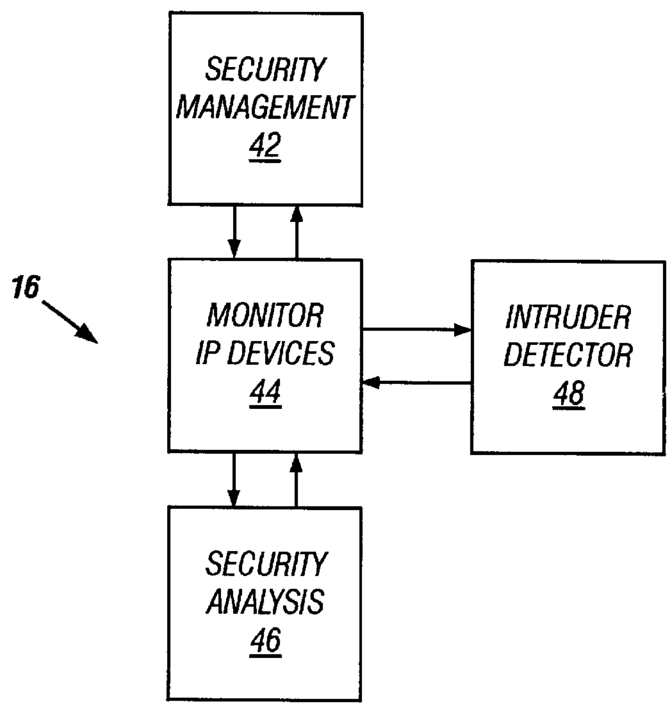

16
SECURITY MANAGEMENT 42
MONITOR IP DEVICES 44
INTRUDER DETECTOR 48
SECURITY ANALYSIS 46

FIG. 2

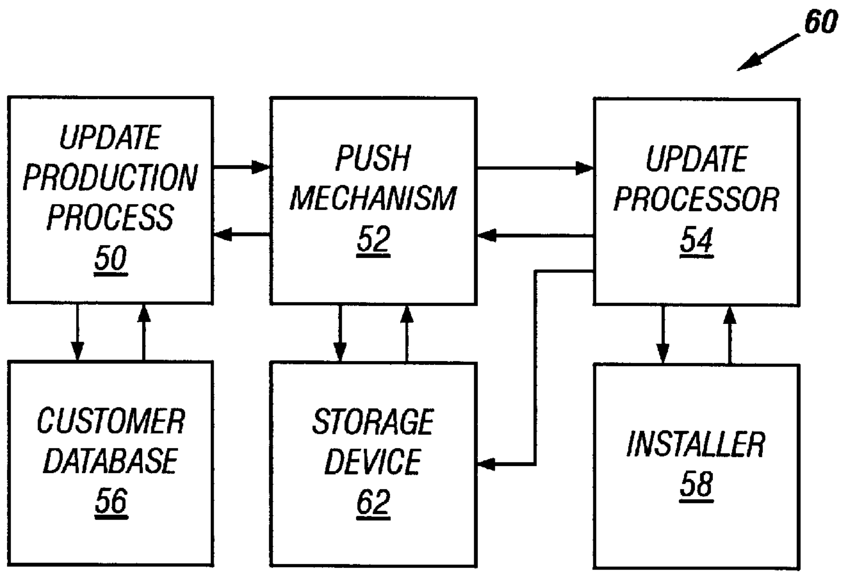

60
UPDATE PRODUCTION PROCESS 50
PUSH MECHANISM 52
UPDATE PROCESSOR 54
CUSTOMER DATABASE 56
STORAGE DEVICE 62
INSTALLER 58

FIG. 3

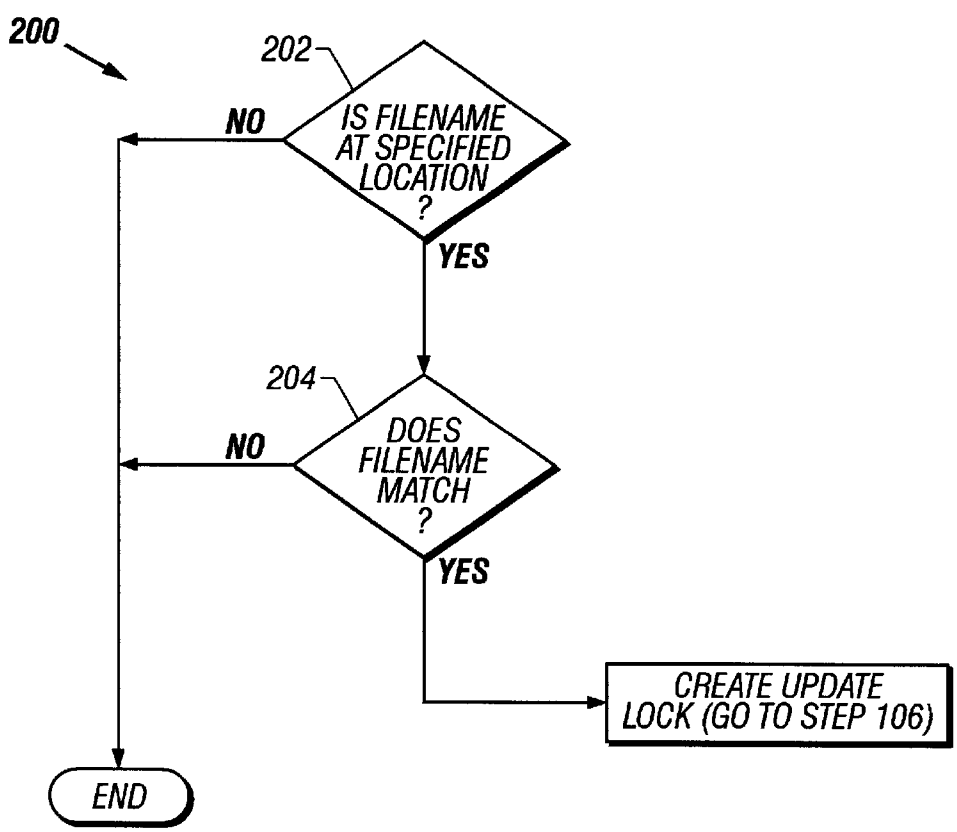

200
202
IS FILENAME AT SPECIFIED LOCATION ?
NO
YES
204
DOES FILENAME MATCH ?
NO
YES
CREATE UPDATE LOCK (GO TO STEP 106)
END

FIG. 4B

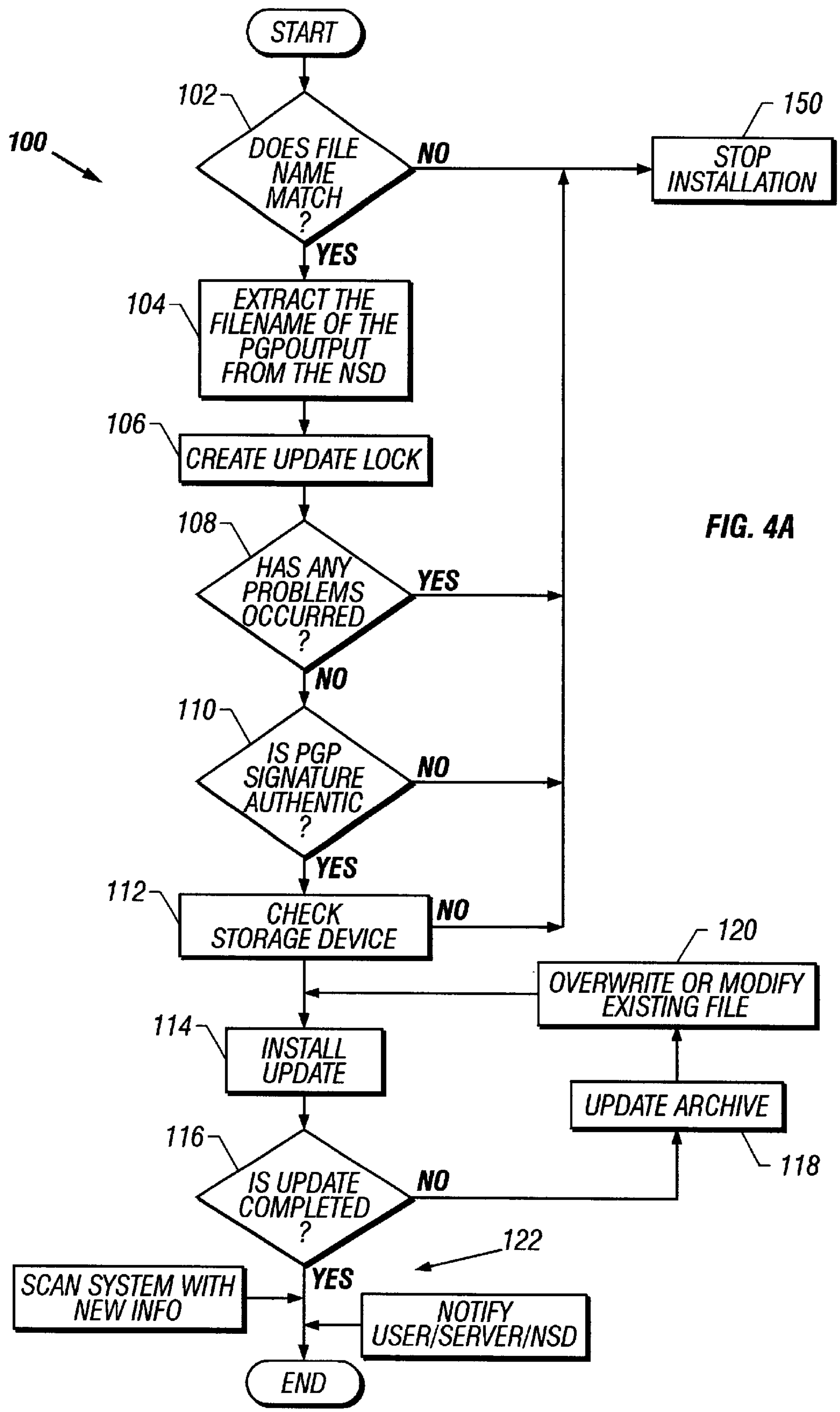

START
102
DOES FILE NAME MATCH ?
NO
YES
150
STOP INSTALLATION
100
104
EXTRACT THE FILENAME OF THE PGPOUTPUT FROM THE NSD
106
CREATE UPDATE LOCK
108
HAS ANY PROBLEMS OCCURRED ?
YES
NO
110
IS PGP SIGNATURE AUTHENTIC ?
NO
YES
112
CHECK STORAGE DEVICE
NO
120
OVERWRITE OR MODIFY EXISTING FILE
114
INSTALL UPDATE
UPDATE ARCHIVE
118
116
IS UPDATE COMPLETED ?
NO
122
YES
SCAN SYSTEM WITH NEW INFO
NOTIFY USER/SERVER/NSD
END

FIG. 4A

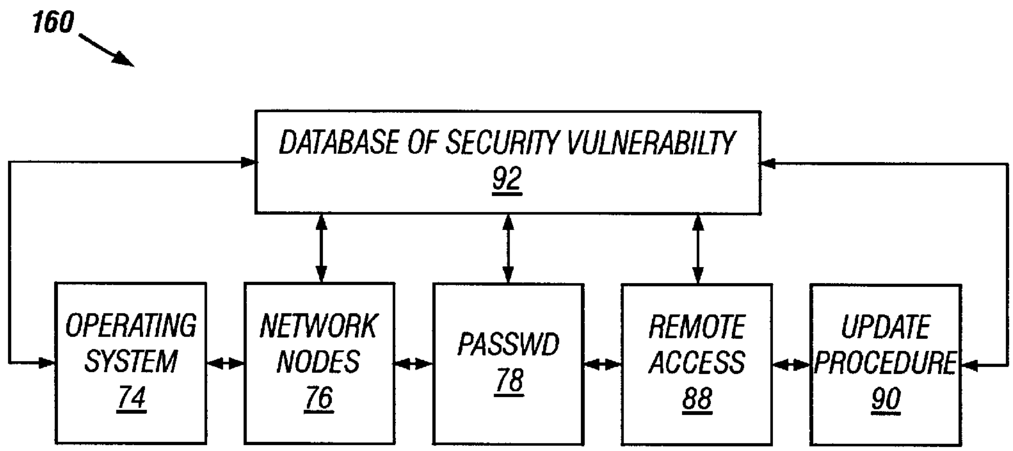

160
DATABASE OF SECURITY VULNERABILTY
92
OPERATING SYSTEM
74
NETWORK NODES
76
PASSWD
78
REMOTE ACCESS
88
UPDATE PROCEDURE
90

FIG. 5

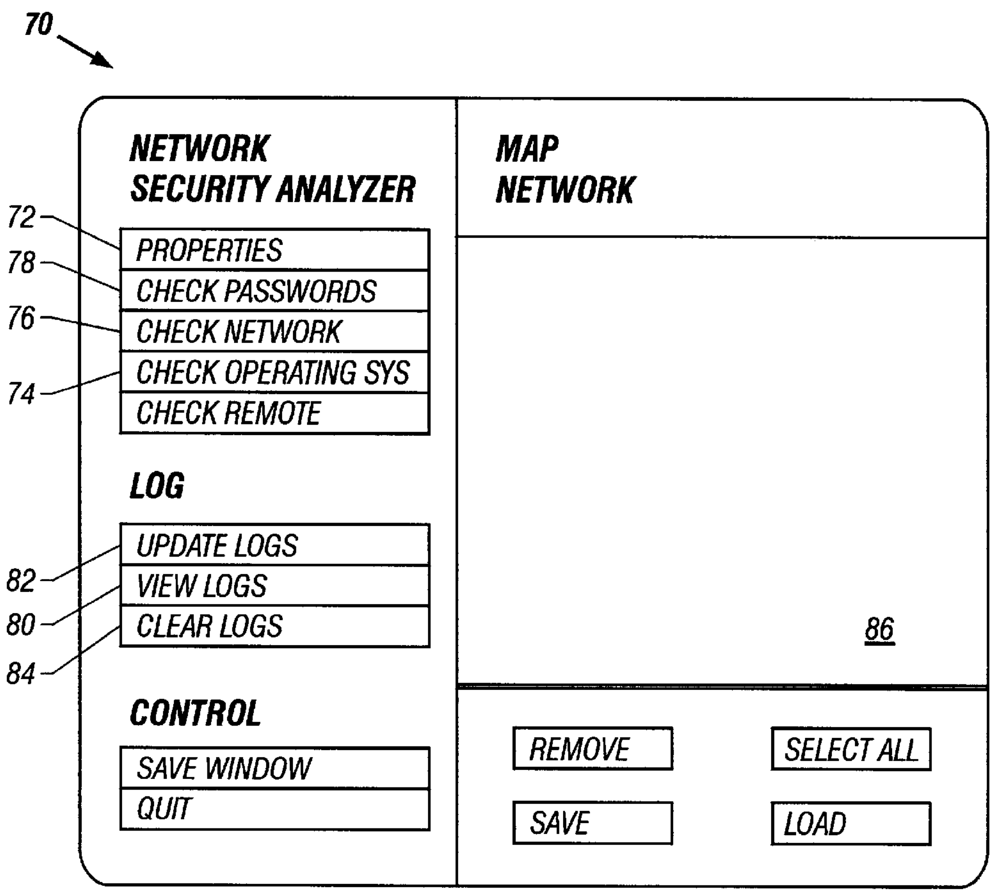

70
NETWORK SECURITY ANALYZER
PROPERTIES
CHECK PASSWORDS
CHECK NETWORK
CHECK OPERATING SYS
CHECK REMOTE
LOG
UPDATE LOGS
VIEW LOGS
CLEAR LOGS
CONTROL
SAVE WINDOW
QUIT
MAP NETWORK
86
REMOVE
SELECT ALL
SAVE
LOAD
72
78
76
74
82
80
84

FIG. 6

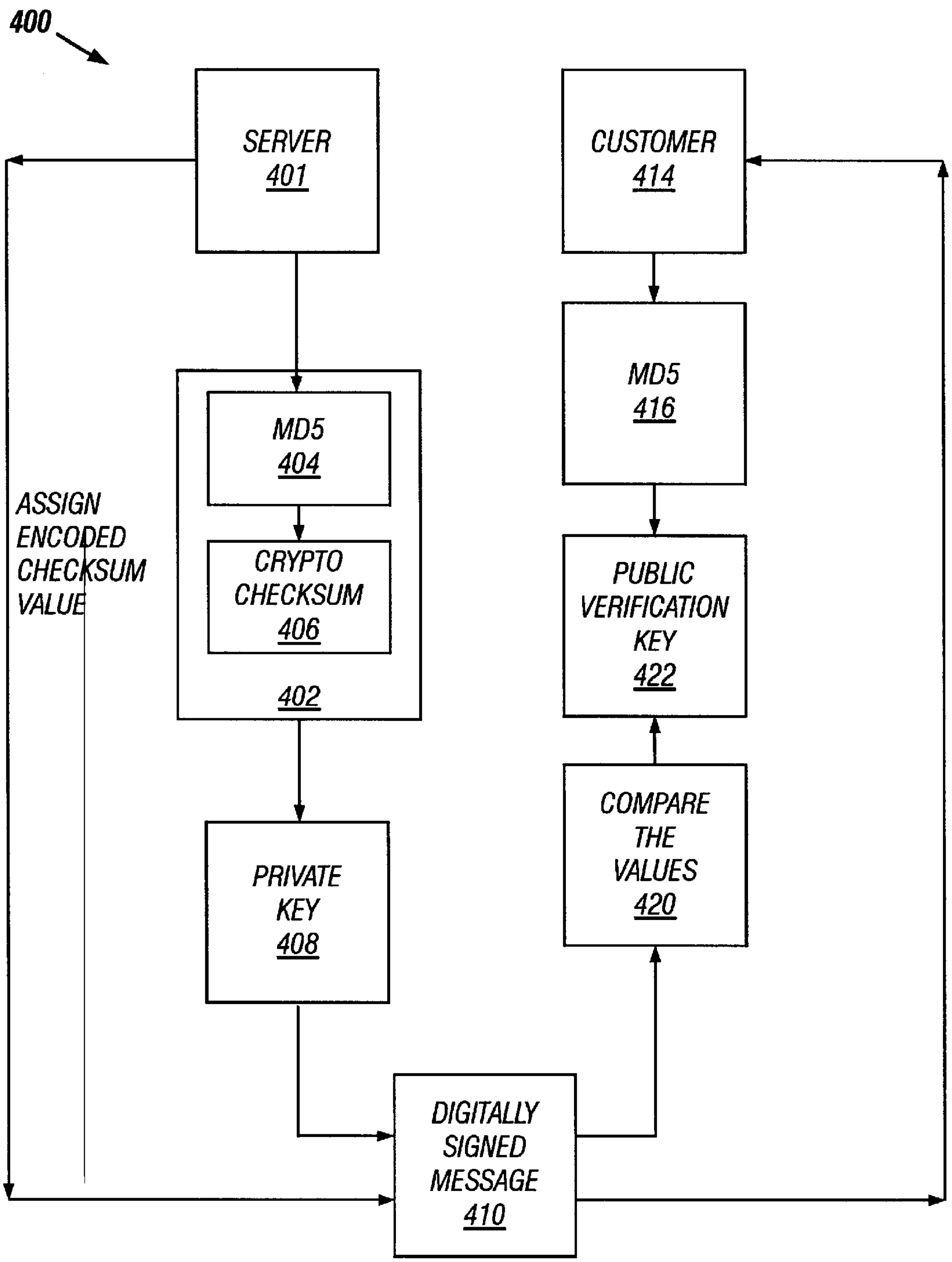

400
SERVER
401
CUSTOMER
414
MD5
416
MD5
404
CRYPTO
CHECKSUM
406
402
ASSIGN
ENCODED
CHECKSUM
VALUE
PUBLIC
VERIFICATION
KEY
422
COMPARE
THE
VALUES
420
PRIVATE
KEY
408
DIGITALLY
SIGNED
MESSAGE
410

FIG. 7

COMPUTER SECURITY

TECHNICAL FIELD

This invention relates generally to computer security software and systems.

BACKGROUND INFORMATION

The rapid development of intranets, extranets and the internet has introduced an increased level of security problems for network managers, computer information systems professionals, individual users, and corporations with an expanding base of telecommuters. With the advent of electronic mail and electronic commerce via the internet, computer information security is an increasing worldwide concern. The responsibilities of system administrators to provide and monitor network connections for security breaches has substantially increased. Furthermore, with the rapid increase of new computer users and the constant development of sophisticated techniques for breaching established network security systems, system administrators are unable to provide their clients and servers with adequate protection. As a result, computer network systems have become increasingly vulnerable to attacks.

In an attempt to prevent unwanted access to computer networks, systems administrators have employed various techniques. One such technique employs a firewall to protect the network clients and servers. A firewall is a screen between a user external to the network and the network and is usually the first line of defense against unauthorized users seeking access to a network. The firewall behaves much like an electronic filter that determines whether a particular user has the requisite security clearance to gain access to the network or computer. As an initial defense, the firewall generally provides adequate protection. However, depending upon the concentration of network traffic, quality of the firewall, and the sophistication, skill and motivation of the person seeking access, the firewall becomes vulnerable to attack. Furthermore, firewalls are designed to prevent unauthorized external access and do not prevent internal users from breaching network security.

In addition, there are products available in the public domain directed to uncovering security vulnerabilities within networks. Although the software tools are not explicitly designed for use by hackers, the tools may be used to gain unauthorized access to a network. For example, a software tool that is widely available is the system administrator tool for analyzing networks (SATAN). This software tool may be used to probe for security holes within a network and highlight network vulnerabilities. An intruder is then able take advantage of the information obtained from SATAN to gain unauthorized access to a network.

The list of network vulnerabilities is always changing and usually well known by hackers. Over the years, hackers have developed many techniques for breaching computer security. Many of the techniques often involve exploiting the vulnerabilities associated with particular software packages. For example, hackers are aware of vulnerabilities in software programs like electronic mail (e-mail), software features like remote login (rlogin), or security weaknesses in particular word processing programs, and they use this information to gain unauthorized access to a network or computer.

One technique used by hackers to breach computer network security is Internet Protocol spoofing (IP spoofing). Using this technique, an unauthorized user gains access to a network by hiding their true location and masking their Internet Protocol (IP) address or root address. In doing so, the IP address appears acceptable to a network server and the unauthorized user is granted access to the network.

Another known method for breaching network security is the buffer overflow technique. Hackers use this technique to gain access to a network through insecure implementation of a file in a file transfer protocol (FTP) server, an electronic mail system, a network file server (NFS), or through a common gateway interface (CGI). The buffer is essentially a temporary holding place in memory with a fixed size for processing computer programs and a hacker may cause too much information to be placed in a buffer. When the buffer is beyond its capacity, an overflow occurs. The overflow is then sent to another part of memory within a server. The hacker is then able to gain privileged access to the computer from inside the new location in memory, and as a result, security is breached.

Whenever an unauthorized user breaches network security and is allowed free access to the system, the damage that might result is unpredictable. However, because some of the system vulnerabilities and techniques used by hackers are known, a system administrator may use that information to make the network less vulnerable to attack. However, the system administrator is required to remain constantly vigilant as to the new attacks being used by hackers, and then use that information to protect the network, clients and servers from the newly found vulnerability.

SUMMARY OF THE INVENTION

While some system administrators may be equipped with software packages that assist them in providing security for their networks, updates to those software packages typically are not automatically provided in real-time, nor are they provided as soon as a new vulnerability is discovered. One aspect of the present invention is that it automatically provides, in real-time, software enhancements with-updated information regarding security vulnerabilities. Thus, a user, system administrator, server, etc. is able to implement prevention techniques before a security breach occurs. In accordance with this aspect of the invention, the enhancement that was sent is then integrated into the computer security software. Before the integration, a computer check can be performed to determine the integrity and the authenticity of the enhancement. The computer check can use cryptographic techniques such as digital signatures and Pretty Good Privacy™ (PGP™) encryption.

In one aspect, the invention provides the most recent information regarding new security attacks. A user can either request the enhancement, or it can be automatically sent (e.g., via the internet) when it becomes available. The software enhancement can include a new version of the software and an update to a database of known security vulnerabilities. A user thus can obtain instant access to the latest security vulnerabilities and employ immediate remedial action before a security breach occurs. Thus, systems and methods according to the invention are not bounded by a static database of security vulnerabilities information. The present invention obviates the need to manually update a computer security system.

In another aspect, the invention relates to a network security detector that is used to monitor security intrusions on a network. The network security detector (NSD) may consist of a single software application dedicated to continuously scanning the network. However, in the disclosed invention the NSD consists of a first application that provides real-time intrusion detection; a second application that

behaves like a system manager; a third application that is able to simulate attacks on the network and monitor Internet Protocol devices; a fourth application that performs a comprehensive security assessment of the network; and a fifth application responsible for receiving the software enhancements.

In another aspect, the invention relates to an integrated system for assessing vulnerabilities. The integrated system includes a database of security vulnerabilities and various modules. A first module accesses the database and assesses security vulnerabilities of an operating system of a computer. A second module accesses the database and assesses security vulnerabilities of a computer network that includes the computer. A third module accesses the database and assesses security vulnerabilities in passwords used to access the computer or the network. A fourth module accesses the database and assesses security vulnerabilities of a remote computer connected to the network. A fifth module receives an update to the database and updates the database. A sixth module is a communications module that allows communication between the integrated system and a similar system.

In yet another aspect of the invention, the invention involves an integrated system for assessing vulnerabilities, including a first module for assessing security vulnerabilities of an operating system of a computer, and a second module for assessing security vulnerabilities of a computer network that includes the computer. The system can also include a database of security vulnerabilities, a third module for accessing the database and for assessing security vulnerabilities in passwords used to access the computer or the network, a fourth module for accessing the database and for assessing security vulnerabilities of a remote computer connected to the network, and a fifth module for receiving an update to the database and updating the database.

The foregoing and other objects, aspects, features, and advantages of the invention will become more apparent from the following description and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings, like reference characters generally refer to the same parts throughout the different views. Also, the drawings are not necessarily to scale, emphasis instead generally being placed upon illustrating the principles of the invention.

FIG. **1** is a schematic diagram of a computer network.

FIG. **2** is a schematic diagram of the network security detector for providing security on a local area network.

FIG. **3** is a schematic diagram of a "push" system for delivering enhancements to a computer security system.

FIG. **4**A is a flow chart of a procedure for installing enhancements to a security vulnerabilities database.

FIG. **4**B is a flow chart of a procedure for extracting the filename of a security vulnerabilities database.

FIG. **5** is schematic diagram of the integrated security system modules for assessing security vulnerabilities on a computer network.

FIG. **6** is a graphical user interface that shows some of the functions presented to a user on a computer monitor, each of the functions relating to assessing security vulnerabilities on a computer and/or a computer network.

FIG. **7** is a flow chart of a Pretty Good Privacy™ (PGP™) authentication procedure for checking the integrity and authenticity of a software enhancement or update.

DESCRIPTION

In accordance with the invention, a database of security vulnerabilities is automatically updated via an electronic network. The database is part of a computer security software system. The automatic update can occur whenever a software enhancement becomes available. The update can then be integrated into the computer security software. New and different security vulnerabilities are discovered almost daily. As a result, computer security checks should employ a flexible mechanism able to adapt to newly discovered security vulnerabilities. The present invention provides such a mechanism by automatically providing enhancements to a database of security vulnerabilities and using that information to provide security solutions to potentially "weak" computer networks and/or computers.

Referring to FIG. **1**, a network includes a local network environment **10**, a data bus **14** for electronically linking various ports of the network, a local server **18**, a network security detector (NSD) **16**, and a firewall **12** for screening unauthorized external users **8**. The local network environment **10** includes connections to the internet and routers for connecting authorized remote locations. System administrators can create a secure environment by using the firewall **12** and supplementing it with the network security detector **16**.

The firewall **12** is an electronic filter used to prevent unauthorized external users **8** from accessing the network **20** without permission. The network security detector **16** ascertains whether unauthorized external users **8** and authorized local users **6** possess the requisite security clearance to access certain areas within the network **20**. The network security detector **16** can be used to prevent both unauthorized external users **8** and authorized local users **6** from unauthorized access to the local server **18**. The firewall **12** and/or the NSD **16** may be subject to attack.

The network security detector **16** can be electrically connected to a database of security vulnerabilities which may be stored on the local server **18**. However, in another embodiment of the invention, a database of security vulnerabilities is stored on the individual computers of the authorized local users **6** or at a remote location not local to the network **20**. The database of security vulnerabilities includes a list of techniques used by hackers to gain unauthorized access to the network **20** and includes a catalog of known security weaknesses in software programs stored on the network **20**. In one embodiment of the invention, the database of security vulnerabilities is used in conjunction with the network security detector **16** to provide security for the network **20**.

As previously mentioned, the list of computer and network vulnerabilities is always changing and growing, and over the years, hackers have developed many techniques for breaching computer security. Table 1 is a list of some of the known features that hackers have used to gain access to computer networks. Table 1 lists common vulnerabilities found on the network **20** that might be exploited. Although the firewall **12** is normally the first line of defense against an attack on the network **20**, the firewall **12** can be circumvented using many techniques, such as IP spoofing and other types of attacks discussed below.

Table 1 shows that features often provided for the convenience of authorized network users may be exploited by hackers or unauthorized external users **8** to access the network **20**. For example, in UNIX™, finger is a feature that allows authorized local users **6** to locate other users on the network **20**, or netstat is used to obtain information regarding network status. However, an individual motivated to breach network security may use these features to gather information about valid users (such as internet addresses) and then use that information to gain unauthorized access to

the network **20**. Hackers may also gain unauthorized access by using programs stored on the network **20** like sendmail and X-windows™ that may allow access to program libraries or give out too much information about authorized network users.

Another avenue onto the network **20** is through a daemon which is a program intended to provide useful services that is not explicitly invoked but lies dormant waiting for some condition on the network **20** to occur. The idea is that the perpetrator of the condition need not be aware that a daemon is lurking although a program may commit an action only to invoke the daemon. For example, printing a file may first invoke the daemon for spooling and then print the file. Another daemon is the hypertext transfer protocol daemon (HTTPD) which is a program used to provide information for the world wide web. However, unless configured properly, the path through an HTTPD may also allow unauthorized external users **8** through the firewall **12** and onto the network **20**.

Depending upon the motivation of the unauthorized external user **8**, the attack may consist of placing a disruption on the system that limits access and not necessarily removing or copying secret documents. Methods for limiting processing on the network **20** are called denial of service attacks. Some techniques used to deny service is called the teardrop and land attack where a hacker sends pairs of carefully constructed IP fragments to a network server. The IP fragments trigger bugs in computer programs or network logic. The overlapping offsets cause the second packet to overwrite data in the middle of the user datagram protocol (UDP) header in such a way that the datagrams are left incomplete. When a software program then reads the invalid datagrams, the program allocates kernel memory, and if enough of the invalid datagrams are received, then the software program is indefinitely suspended.

There are several other commonly used attacks that produce the aforementioned result namely UDP bombs, ping floods, and SYN floods. SYN represents the synchronizing bit that indicates to a server that a client is seeking access. The SYN flood attack bombards a system with dozens of falsified connection requests a minute and can seriously degrade a system's ability to give service to legitimate connection requests. Accordingly, the attack is said to "deny service" to system users. In addition, a hacker may use some of the preceding techniques to store programs on the network **20** that could be used to gain access at a later time. That is, the hacker builds a backdoor or trap door onto the network **20** that might go undetected, and as a result this enables the hacker to exploit the network **20** at any given time.

TABLE 1

An example of the type of information contained in a security vulnerabilities database.

Feature	Vulnerability
Firewall	Check the firewall for vulnerability to routing, IP spoofing, and other attacks.
Information Gathering	Check for finger, rusers, netstat, and many other sources of network information that is useful during an attack.
Sendmail	Check for historical vulnerabilities in sendmail and misconfigurations that may give out too much information or allow outsiders in. Also check for buffer overflows that allow local users to gain root.
File Transfer Protocol (FTP)	Check for world-writable directories, insecure files inside the FTP sandbox, and other attacks that may crash the server, plant backdoors into the system, or allow users to escape the FTP sandbox.
Network File Server	Check for insecure shares, filehandle guessing, and other common misconfigurations that allow outsiders to see the disks or create backdoors to the system.
HyperText Transfer Protocol Daemon and Internet Information Server (HTTPD and IIS)	Check for known bugs in the servers, commonly present CGI scripts that are vulnerable to buffer overflow attacks, server misconfigurations, and other vulnerabilities that allow outsiders to escape the server sandbox, gain root or user access to the system, or crash the server or system.
Miscellaneous Daemon	Check for the presence of network services that are inherently insecure, such as telnetd, walld, tfpd, and many others.
X-windows ®	Check for open permissions that allow snooping of remote X sessions, unpatched libraries and executables vulnerable to buffer overflow attacks, and other well-known X vulnerabilities.
Denial of Service Attacks	Check for vulnerability to all common attacks SYN Floods, UDP Bombs, Ping Floods, and others, plus newer attacks like Land and Teardrop.
Home	Checks home directory for world writeability.
System	Checks for read/write files in the configuration files.
Password	Checks for /etc/passwd, easy guessable passwords.
Configuration Files Check	Look for common misconfigurations in various files like inetd.conf, .rhosts, host.equiv, and many others.
Users	Checks for home directory important files with insecure permissions and pays special attention to root's files.
Backdoors	Checks for possible backdoors in the system binaries and configuration files.
Patches	Checks for the presence of all Sun ® security.

The Network Security Detector (NSD)

As previously mentioned, the firewall **12** is often the first line of defense against some of the aforementioned security attacks on the network **20**. However, according to the invention, another layer of defense involves the use of a security network detector **16**. In one embodiment of the invention, the network security detector **16** includes software programs that seek to uncover security intrusions. In one embodiment of the invention, the network security detector **16** is a single package that continuously scans the network for violators. In another embodiment of the invention, the network security detector **16** is an integrated family of software packages that individually resolve various security issues. Referring to FIG. **2**, the network security detector **16** has various components. In one embodiment of the network security detector **16**, it has at least four integrated software applications for providing network security.

A first application **48** of the NSD **16** provides a real-time intrusion detection notification system. In one embodiment, the first application **48** takes an action which may include sending an alarm to a system administrator if an intrusion is detected. In addition, the first application **48** electronically disengages the intruder and marks the intruder's location. The first application **48** can also distribute to each computer on the network **20** information about network status. The first application **48** can watch each site location and electronically communicate with a system manager, or his programmatic agent. The system manager can be a system administrator. However, the system manager could be another software program in electrical communication with the first application **48**.

A second application **42** of the NSD **16** can include the system manager for receiving information from the first

application **48**. In this capacity, the system manager assists the system administrator in providing network security information and assists in managing the security of numerous port connections associated with the network **20**.

A third application **44** of the NSD **16** can provide continuous monitoring of the complete network **20**. The third application **44** is in electronic communication with the other applications and may be used as a system management tool. However, in the disclosed embodiment, the third application **44** is used to monitor Internet Protocol devices.

In another aspect of the invention, the third application **44** can also simulate an attack on the network **20**. The simulation can provide information to the network security detector **16** for uncovering potential security vulnerabilities before the vulnerabilities are exploited. The third application **44** may also check the local server **18** for security vulnerabilities. In addition, the third application **44** provides a map of all ports on the network **20** and pings all Internet Protocol devices to expose potential security vulnerabilities.

A fourth application **46** of the NSD **16** can perform a comprehensive security assessment of the network **20**. The fourth application assesses the operating system of various computers and monitors the network for security vulnerabilities. The fourth application **46** can also provide a report of all security breaches and provide an appropriate solution based on a database of known security vulnerabilities similar to Table 1. However, because vulnerabilities are constantly changing and new ones are being discovered, an efficient means is required to update the database of security vulnerabilities.

A fifth application is responsible for receiving the software enhancements and updating of the database of security vulnerabilities. The following is a discussion of how enhancements are automatically provided immediately and in real-time for a computer security vulnerabilities database.

The Push System

Because network security vulnerabilities are constantly changing and new ones being developed by hackers, a system administrator is required to remain vigilant in order to protect the computer network, clients, and servers from the new system vulnerabilities. The disclosed invention provides a means for obtaining real-time enhancements to a database of security vulnerabilities. In this way, a system administrator or a dedicated network server is able to take immediate action to protect the computer network, clients and servers before a breach occurs. The push system is a method for automatically providing software enhancements in real-time, and is an efficient means for providing enhancements to a database of security vulnerabilities. As described below, the push system is a part of an integrated security system that primarily provides a secure network operating environment. Specifically, the push system is part of a fifth module that receives an update and updates a database of security vulnerabilities.

In one embodiment of the invention, the push system provides computer security software enhancements for execution on at least one computer. The push system automatically implements and electronically sends computer software enhancements over a computer network when the software enhancement becomes available. The software enhancement can include an update to a computer security vulnerabilities database or a new version of an entire computer security software package. In either embodiment of the invention, the software enhancement is automatically distributed over an electronic network.

In an alternative embodiment, the push system is manually activated by a user seeking an update. In this alternate embodiment, the user is able to send a query to a server about the availability of an enhancement which can include an update of the database or a new version. If the enhancement is available, the server either pushes the enhancement over the network to the user or provides a negative response if it cannot push the enhancement for some reason.

The software update can include a combination of old and new information regarding computer security vulnerabilities for inclusion in the database, where succeeding updates primarily add new information to the database. As a result, the information in the database continuously builds. However, if the enhancement is a new version of the computer security software, the new version includes not only the new database of information but also includes new features or functionalities not a part of the original software. Thus, a new version might require overwriting the old database of information or discarding the old version and re-installing a new database.

The push system integrates the software enhancement into existing programs. Additionally, the integration can also perform a check on the integrity and authenticity of the software enhancement provided. This feature determines whether the user being sent the software enhancement is eligible, and checks the integrity and authenticity of the software enhancement. In determining the integrity and authenticity of the software enhancement, the push system can use digital signatures or other cryptographic techniques. In the disclosed embodiment, digital signatures are used to encode the software enhancement by using a signing key, and an authorized local user **6** or customer possesses the correct key for validating the original message.

The push system also installs the software enhancements. The installation includes performing a check on the software enhancement and determining the integrity and authenticity of the software enhancement. Referring to FIG. **3**, the push system **60** includes three primary components: an update production process **50**; a push mechanism **52**; and an update processor **54**. The push system **60** also includes a customer database **56** and an installer **58**. The following is a description of each component.

The Push Mechanism

Still referring to FIG. **3**, the push mechanism **52** delivers the software enhancement to the customer and invokes an installer **58** via the update processor **54**. The push mechanism **52** delivers the software enhancement using electronic mail to a small script which places the contents of the push onto a storage device **62**. The script is a program written in a high level computer language that a local server **18** may execute. The script can include different commands and subroutines for accessing software applications from various memory locations within the computer. The script may be used to implement storing an update of a database of security vulnerabilities **92** on a storage device **62** and then automatically running an update installation procedure **100**.

In the disclosed embodiment, the push mechanism **52** is invoked using an electronic mail message system that is delivered using simple mail transfer protocol (SMTP) which is a standard method for transmitting electronic mail to and from the internet. However, the push mechanism may also use a post office protocol (POP) mail server. As is well known in the art, the POP server behaves much like a post office box. The POP server holds mail for the user, and when the user connects to the POP server, their mail is automatically transferred to them. Thus, in this particular example, the authorized local user **6** could connect to the POP server and the security vulnerabilities enhancement would automatically be pushed onto his computer, server or network.

In another embodiment of the invention, the security vulnerabilities database is delivered using an internet message access protocol (IMAP) mail service. IMAP offers a built-in flexibility that enables an authorized local user **6** to access electronic mail messages from a stand alone computer, workstation, or a laptop computer without transferring files between the computers. Thus a client is able to access remote messages as if they were on a local server **18**.

Techniques For Implementing Data Transfer

In a client-server configuration of the network **20**, the software enhancements are stored on a server, and the server is able to distribute the software enhancements to a client using a of techniques. In the disclosed invention, the server is remotely connected to the client's network **20**. Using the push mechanism **52** as described above, the remote server is able to initiate contact with a client. Client information is obtained from the customer database **56**. When the software enhancements become available, the push mechanism **52** is invoked. The push mechanism **52** takes the contents of a specified location within the remote server and sends the software enhancement to the client via electronic mail. As a result, whenever the software enhancement becomes available, it is immediately pushed over a computer network to the client

In another aspect of the invention, the client may initiate contact with the remote server by first inquiring whether the software enhancement is available. If the software enhancement is available, that is if there is new information at a specified memory location, then the client is able to receive delivery of the software enhancement. As a result, the client has performed a pull from the server in order to obtain the software enhancements. In another aspect of the invention, the client or another machine acting on behalf of the client may constantly interrogate the remote server about the status of a software enhancement. Similarly, when the software enhancement becomes available it is immediately delivered to the client.

In another embodiment of the invention, the software enhancements are provided using a file transfer protocol (FTP) program. As is well known in the art, FTP allows the direct transfer of files across a computer network. However, unlike the push or pull mechanisms described above, performing an FTP may involve a variety of proxies to facilitate the file transfer. In addition, FTP as a push mechanism would require allowing access to the portion of memory where the software enhancement transfer files are located which would create a network security vulnerability.

In another aspect of the invention, the software enhancements may be automatically sent to an authorized local user **6** or client on a diskette or on a compact disk read only memory (CD-ROM) storage device. However, this technique minimizes being able to immediately obtain enhancements when they become available, and presents logistical problems that should be avoided.

The Update Production Process

In the disclosed invention, the update production process **50** is dedicated to tracking vulnerabilities and maintaining a database of security vulnerabilities. In one aspect of the invention, the update production process **50** identifies and prioritizes the vulnerability, specifies the type of attack, archives the attack source code, creates a report of the vulnerability, and integrates the new attack into a library of known vulnerabilities. The vulnerabilities are tracked from a variety of sources. The sources include mailing lists, internet web sites and information disseminated by hackers. When new vulnerabilities are discovered, they are classified and prioritized. In the disclosed invention, the vulnerabilities are prioritized based on the type of attack using a numerical range from one to ten. The prioritization is based upon the source of the information, the potential damage that the new vulnerability might produce, or the type of attack to which the vulnerability is targeted (e.g., the network, a local attack, or the operating system). The integration of the new vulnerabilities also includes placing the database of known security vulnerabilities in communication with the network security detector **16**.

The Installation Procedure

In the disclosed invention, the update processor **54** installs the software enhancement when the enhancement is received by the customer. Referring to FIG. **4**A, the installation procedure **100** is a means for processing the software enhancement. Note during each step of the installation procedure **100**, that if any step fails (Step **108**), then the installation procedure **100** stops further installation (Step **150**). When the enhancement arrives at an authorized local user's **6** location, the installation procedure **100** installs the software enhancement onto a storage device **62** and performs a series of reliability checks. Prior to installing the software enhancement on a computer or on a local server **18**, the authenticity and integrity of the software enhancement is determined. The authenticity checks may occur either at the user's computer or at the local server **18**. The authenticity checks include performing a cryptographic technique by verifying digital signatures, authenticating the software, and verifying the user before installing the software enhancement. The installer **58** is always invoked on a temporary disk file and contains no programming code (e.g., hyper-text markup language (HTML) or channel definition format (CDF), etc.). As a result, this feature allows for optimal push channel independence. That is, because the installer **58** can read and receive the software enhancements in any format and from any location, the installer **58** is push channel independent. The installer **58** is usually invoked on a temporary file which contains no network markup from transport layers, such as HTTP or CDF. This method allows for independence from any particular push channel. Alternatively, details regarding push channel independence may be included as processing information.

The installer **58** begins the installation procedure **100** by first ensuring that the filename of the software enhancement matches a predetermined string (Step **102**). In another embodiment of the invention, the filename is represented by a string of variables and numbers appended by a suffix that indicates the type of document. The installer **58** then checks for a specific location to extract the filename (Step **104**) to a PGP™ output.

Referring to FIG. **4**B, the installer **58** performs a filename extraction sequence **200**. The filename extraction sequence **200** consists of determining whether the filename of the software enhancement is at a certain location (Step **202**) within the software enhancement. If the filename is at the specified location, then the installer **58** extracts the filename and performs a matching sequence (Step **204**). After the filename is extracted and matched, the installer **58** creates an update lock (Step **106**). Referring to FIG. **4**A, the update lock (Step **106**) disables any other version of the software enhancement being installed from functioning. The installer **58** then checks for the PGP™ digital signature (Step **110**). A digital signature is provided by a software security package residing at a remote location. In the disclosed invention, the remote location is a site residing with the provider of the software enhancement. The remote location includes a computer that maintains all security keys pertaining to the digital signature. As a precautionary measure, that computer is

off-line and electrically isolated from the client-server network. As a result, no new security vulnerabilities are created and an additional level of computer network security is provided.

In one embodiment of the invention, the software security package the provides the digital signature is the network security detector **16**. In the disclosed invention, the security software package uses Pretty Good Privacy™ (PGP™) encryption for authentication and provides a digital signature to facilitate authentication. A digital signature is a cryptographic function computed as a message and a user's private key. The private key is a number or a mathematical value that is unique to the sender. The signature function produces a value unique to the private key and the fingerprint value being signed. The private key has a mathematically related public key that anyone may use to verify the signature created by the private key.

The message that is signed is typically a condensed version of the actual message produced by a message digest (MD) or hash algorithm. In general, a message digest algorithm, takes as an input a message of arbitrary length and produces a shorter fingerprint of the input. In the disclosed invention, the message digest algorithm used is called MD5 and produces a 128-bit fingerprint. The message digest algorithm is generated by a transformation function that produces a fixed size representation of the original message. The message digest function has the properties that it is difficult to predict the value of the function for a given input, and that it is difficult to find two arbitrary messages with the same fingerprint, or given a fingerprint, it is difficult to find a second value for the given fingerprint.

At the receiving end of the message, the recipient verifies the signature on the message using the public key. After the encoded message is sent and properly decoded, the PGP™ authentication process (Step **110**) is completed. The software enhancement package is then sequentially scanned for errors (Step **114**–**122**), and if there is sufficient space (Step **112**) on the storage device, the software enhancement is stored on the client's computer, hard drive or server (Step **114**).

After storing the software enhancement with the client, the installer **58** updates an archive (Step **118**), overwrites any previously existing software enhancement packages (Step **120**), and notifies the client or server that the installation procedure **100** is complete (Steps **122**). When the installation is complete (Step **116**), the installer **58** automatically runs a scan (Step **118**) of the network **20** using the newly installed software enhancement to address any new security vulnerabilities uncovered by the installation procedure **100**. The update processor **54** also includes solutions for repairing the newly discovered vulnerabilities. The update processor **54** may automatically implement the suggested repairs of the system vulnerabilities and may send a message that the update is completed (Step **122**).

In another embodiment of the invention, the installer **58** is able to push source code as a separate enhancement. When the enhancement is received, the source code will not automatically be processed by the update processor **54**. However, the update processor **54** may invoke a source code update mechanism that will prompt the user to install the enhancement source code.

An Integrated Security System

The database of security vulnerabilities is part of an integrated system that provides a secure operating environment. The disclosed invention is an integrated system for assessing computer security vulnerabilities The integrated system includes a database of security vulnerabilities and various modules. A first module accesses the database and assesses security vulnerabilities of an operating system of a computer. A second module accesses the database and assesses security vulnerabilities of a computer network that includes the computer. A third module accesses the database and assesses security vulnerabilities in passwords used to access the computer or the network. A fourth module accesses the database and assesses security vulnerabilities of a remote computer connected to the network. A fifth module receives an update to the database and updates the database. A sixth module is a communications module that allows communication between the integrated security system and a similar system.

Referring to FIGS. **5** and **6**, the various integrated security system modules **160** are represented by corresponding symbols on a graphical user interface (GUI) screen **70**. The first module **74** is used to check the operating system. The check is invoked by using the check operating systems **74'** icon on the GUI screen. The check involves ascertaining whether a user has the correct permission requirements to gain access to the network. Also, in one embodiment of the invention, the first module **74** determines whether all known vulnerabilities have been addressed. Specifically, the first module **74** determines whether the suggested changes resulting from the installation procedure (Step **118**) have been made to the operating system.

The first module **78** uses a binary file integrity checking technique using a message digest number 5 (MD5) checksum for files stored on a disk. In the disclosed embodiment of the invention, the checksum is used to verify that no errors have occurred when reading a particular string of bits or a particular file. The checksum value can be any checksum method where it would be difficult to predict the checksum value for a given input. As described below, no two checksums are equivalent, and any changes or corruption of the stored data is detected. Thus, by using a database of MD5 checksum values, the third module **78** is able to determine whether the software enhancement stored on a storage device **62** was modified after a snapshot had been taken, or after the database was created.

The second module **76** accesses the database of security vulnerabilities **92** and assesses network security. The second module **76** connects to a network service, accepts information from the service and interrogates the service. The second module **76** performs a network scan and may be invoked by activating the check network **76'** icon on the graphical user interface screen. The network scan produces a map of the network **86** which is essentially an inventory of the Internet Protocol (IP) devices connected to the network. Using network protocol, the integrated system also probes the ports of each of the IP devices for programs that contain security vulnerabilities that may be exploited. The network scan ensures that the network **20** and a local server **18** is protected against any unauthorized access that may penetrate the firewall **12**. The network scan for IP devices is invoked using the properties (PROP) icon **72** which enables an authorized local user **6** to configure the various modules.

The third module **78** accesses the database of security vulnerabilities **92** and assess security vulnerabilities in the passwords being used to access a computer or a computer network **20**. The third module **78** uses a dictionary of passwords, common English words, and other words to compare and identify vulnerable passwords. The third module may be invoked by an authorized user **7** by activating the check password **78'** icon on the graphical user interface screen. When invoked, the third module **78** checks whether the words in a list have been used as passwords.

The fourth module **88** accesses the database of security vulnerabilities **92** and assess the security vulnerabilities of a remote computer connected to the network. The fourth module **88** allows a remote computer to first connect to a network service then accepts information from the service and like the second module **76**, also interrogates the service.

The fifth module **90** is for receiving an update to the database of security vulnerabilities **92** and updating the database. As described above, the fifth module **90** includes the installer **58**. The fifth module **90** checks the authenticity and integrity of the software enhancement. The authenticity of the software enhancement works for either an update or a new version. The authenticity and integrity of the software enhancement is confirmed using the previously described cryptographic methods with PGP™ output from the network security detector **16**. In one aspect of the invention, the fifth module **90** also maintains a record of all transactions.

A sixth module is a communications module that allows the integrated security system **160** to communicate with a similar system over a computer network. The sixth module may allow communication between the similar system and the various modules and software applications for sharing database files, for sharing workload in breaking long lists of passwords, transmitting reports or data for purposes of analysis, reporting to a management station, configuring files or configuring an operating system, and invoking a remote system to send a software enhancement. The sixth module may also include cryptographic code for protecting the confidentiality and integrity of the information being transmitted.

The sixth module may be used for authenticating a user and providing a means for reporting various transactions on the network **20**. Specifically, the sixth module may be used to constantly check a user's identification, the integrity of the service connection, and the status of any network processing.

The GUI **70** may also provide a reporting mechanism. The GUI **70** may also include several means for reporting various network transactions. In the disclosed invention, the GUI **70** includes a log view **80** may allow a user to view a text version the update process or log information on a storage device, a log update **82** that generates a report of all security vulnerabilities on the network **20**, and a log clear function **84** that allows a user to erase the log.

Referring to FIG. **7**, the PGP™ authentication procedure **400** is described. The fifth module **90** performs a message content authentication and verifies that the software enhancement received is exactly the same as the message sent. The fifth module **90** may employ a cryptographic checksum called a message authentication code, or by using digital signatures. The techniques may be used to verify where the message originated, the sender, and the receiver. As a result, the fifth module **90** is able to verify that the actual sender of the message is the person or server that the sender in the message claims to be.

The fifth module **90** uses an asymmetric cryptosystem wherein the recipient of the message is assured the validity of the sender. The fifth module **90** uses a key distribution center or a public key to verify the place of origination. Alternatively, the fifth module **90** may verify a party's identity by using biometrics. In general, biometrics is a method of authenticating a person's identity by using an electronic transmission of personal identifying characteristics of either the recipient, sender, or both.

Still referring to FIG. **7**, the PGP™ authentication procedure **400** used by the fifth module **90** is described further. A server **401** obtains the most recent software enhancement and seeks to deliver a secure copy of the software enhancement to a customer **414**. The server **401** includes an MD5 checksum utility program **404**. Using the checksum utility program **404**, the software enhancement file is compressed into a 128-bit cryptographic checksum **406** and given MD5 checksum value **402**. For example, an Intel™ system running DOS or Microsoft Windows™ and an executable file "md5.exe", has a 128-bit MD5 checksum value equal to 374394a3d46c812f5f6db425ad88f57c for the file "md5.exe", Similarly, the software enhancement program is given a MD5 checksum value with a 128-bit representation. As a result, the software enhancement is uniquely marked and the MD5 checksum value **402** is used to distinguish the software enhancement file.

A cryptographic technique is applied to the cryptographic checksum by attaching a private key **408** to the software enhancement file. The private key **408** is a mathematically generated number that is unique to the sender, and is a number that only the sender knows. The software enhancement is then given a digital signature **410** which is a function of the message digest number and the private key **408**. The signed message is then delivered to the customer **414**. The customer **414** applies an MD5 algorithm **416** to the software enhancement delivered to confirm the cryptographic checksum value **402**. The recipient compares **420** the value obtained from the cryptographic checksum with the value obtained by using a (public) verification key **422**. If the values are equivalent, then the original PGP™ message delivered was created by the holder of the private key.

Variations, modifications, and other implementations of what is described herein will occur to those of ordinary skill in the art without departing from the spirit and the scope of the invention as claimed. Accordingly, the invention is to be defined not by the preceding illustrative description but instead by the spirit and scope of the following claims.

What is claimed is:

1. An integrated system for assessing vulnerabilities, comprising:

a database of security vulnerabilities;

a first module for accessing the database and for assessing security vulnerabilities of an operating system of a computer;

a second module for accessing the database and for assessing security vulnerabilities of a computer network that includes the computer;

a third module for accessing the database and for assessing security vulnerabilities in passwords used to access the computer or the network;

a fourth module for accessing the database and for assessing security vulnerabilities of a remote computer connected to the network; and

a fifth module for receiving an update to the database and updating the database.

2. The integrated system of claim **1** wherein the first module determines permissions of the operating system.

3. The integrated system of claim **1** wherein the first module determines whether predetermined changes have been made to the operating system.

4. The integrated system of claim **1** wherein the second module connects to a network service and accepts information from the service.

5. The integrated system of claim **1** wherein the second module connects to a network service and interrogates the service.

6. The integrated system of claim **1** wherein the third module checks whether the words in a list have been used as passwords.

7. The integrated system of claim 1 wherein the fourth module allows the remote computer to connect to a network service and accepts information from the service.

8. The integrated system of claim 1 wherein the fourth module allows the remote computer to connect to a network service and interrogate the service.

9. The integrated system of claim 1 wherein the fifth module also checks the authenticity and integrity of the update.

10. The integrated system of claim 9 wherein the fifth module employs a cryptographic technique to check the authenticity and integrity of the update.

11. The integrated system of claim 10 wherein the cryptographic technique comprises a digital signature.

12. The integrated system of claim 1 wherein the fifth module receives the update after a request is made for the update.

13. The integrated system of claim 1 wherein the fifth module receives the update automatically whenever the update becomes available.

14. The integrated system of claim 1 wherein a sixth module is a communications module for communicating with a similar system.

* * * * *

Exhibit A-5

US006321334B1

(12) **United States Patent**
Jerger et al.

(10) **Patent No.: US 6,321,334 B1**
(45) **Date of Patent: Nov. 20, 2001**

(54) **ADMINISTERING PERMISSIONS ASSOCIATED WITH A SECURITY ZONE IN A COMPUTER SYSTEM SECURITY MODEL**

(75) Inventors: **Michael S. Jerger**, Kirkland; **Jeffrey A. Bisset**, Issaquah; **Craig T. Sinclair**, Redmond; **Michael J. Toutonghi**, Seattle, all of WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/116,514**

(22) Filed: **Jul. 15, 1998**

(51) **Int. Cl.**[7] **G06F 12/24**
(52) **U.S. Cl.** **713/200**; 713/201
(58) **Field of Search** 713/200, 201, 713/202

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,678,041	10/1997	Baker et al. .
5,684,951	11/1997	Goldman et al. .
5,796,942	8/1998	Esbensen .
5,828,893	10/1998	Wied et al. .
5,835,726	11/1998	Schwed et al. .
5,919,247	7/1999	Van Hoff et al. .
5,930,792	7/1999	Polcyn .
5,940,843	8/1999	Zucknovich et al. .
5,958,005	9/1999	Thorne et al. .
5,958,051	9/1999	Renaud et al. .
5,963,142	10/1999	Zinsky et al. .
5,987,611	11/1999	Freund .
5,991,878	11/1999	McDonough et al. .
6,041,412 *	3/2000	Timson et al. 713/200
6,138,238 *	10/2000	Scheifler et al. 713/200

OTHER PUBLICATIONS

S.M. Bellovin et al., "Network Firewalls," *IEEE Communications Magazine*, Sep. 1994, pp. 50–57.

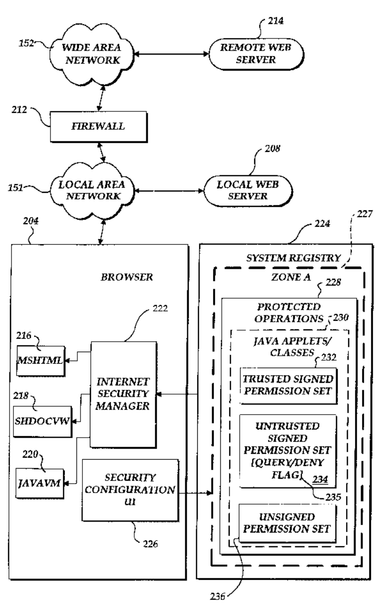
* cited by examiner

Primary Examiner—Ly V. Hua
(74) *Attorney, Agent, or Firm*—Christensen O'Connor Johnson Kindness PLLC

(57) **ABSTRACT**

Computer-based systems and methods are disclosed for a comprehensive security model for managing foreign content downloaded from a computer network. The methods and systems include the configuration of a system security policy that is stored on a host computer. The system security policy includes one or more independently configurable security zones. Each security zone corresponds to a group of network locations and may have one or more associated configurable protected operations that control the access to the host system by foreign content downloaded from the computer network. A protected operations may have one or more associated configurable permissions that define the capabilities of the protected operation. Each permission may be defined by one or more parameters and each parameter may be defined by one or more primitives. The permissions may be defined to enable the permission, disable the permission, or prompt the user when the permission is required. The permission may also be configured to the "fine grained" level of the primitives. Default permission levels that provide predefined parameter and primitive entries that are grouped as high security, medium security, and low security may be selected by the user at most levels of the configuration.

109 Claims, 82 Drawing Sheets



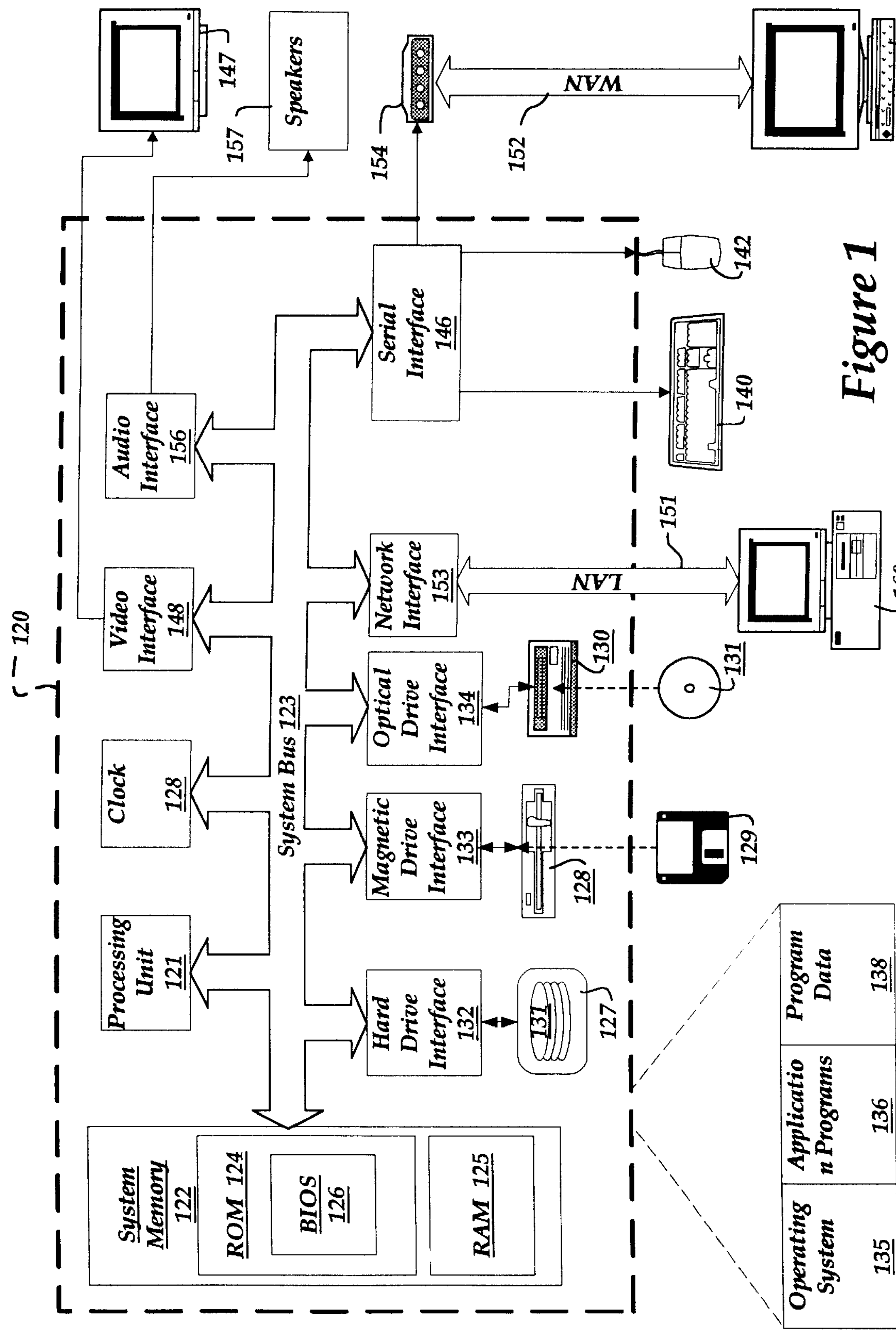

120
System Memory 122
ROM 124
BIOS 126
RAM 125
Processing Unit 121
Clock 128
Video Interface 148
Audio Interface 156
System Bus 123
Hard Drive Interface 132
Magnetic Drive Interface 133
Optical Drive Interface 134
Network Interface 153
Serial Interface 146
131
127
128
129
130
131
LAN
151
160
140
142
147
Speakers
157
154
WAN
152
149
Operating System 135
Application Programs 136
Program Data 138

Figure 1

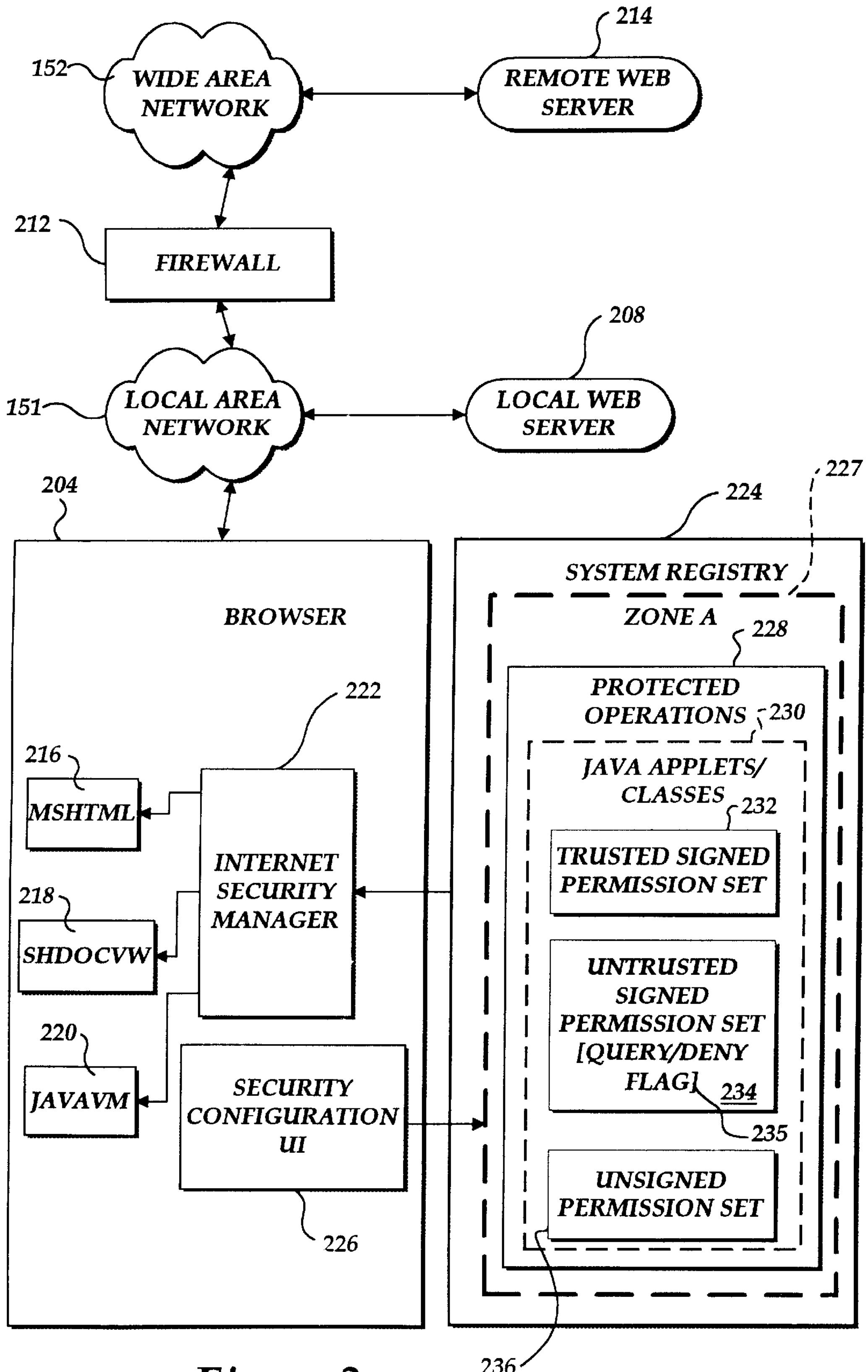

152
WIDE AREA NETWORK
214
REMOTE WEB SERVER
212
FIREWALL
208
151
LOCAL AREA NETWORK
LOCAL WEB SERVER
204
224
227
SYSTEM REGISTRY
BROWSER
ZONE A
228
PROTECTED OPERATIONS
230
222
216
MSHTML
JAVA APPLETS/ CLASSES
232
INTERNET SECURITY MANAGER
TRUSTED SIGNED PERMISSION SET
218
SHDOCVW
UNTRUSTED SIGNED PERMISSION SET [QUERY/DENY FLAG]
234
220
JAVAVM
SECURITY CONFIGURATION UI
235
UNSIGNED PERMISSION SET
226
236

Figure 2

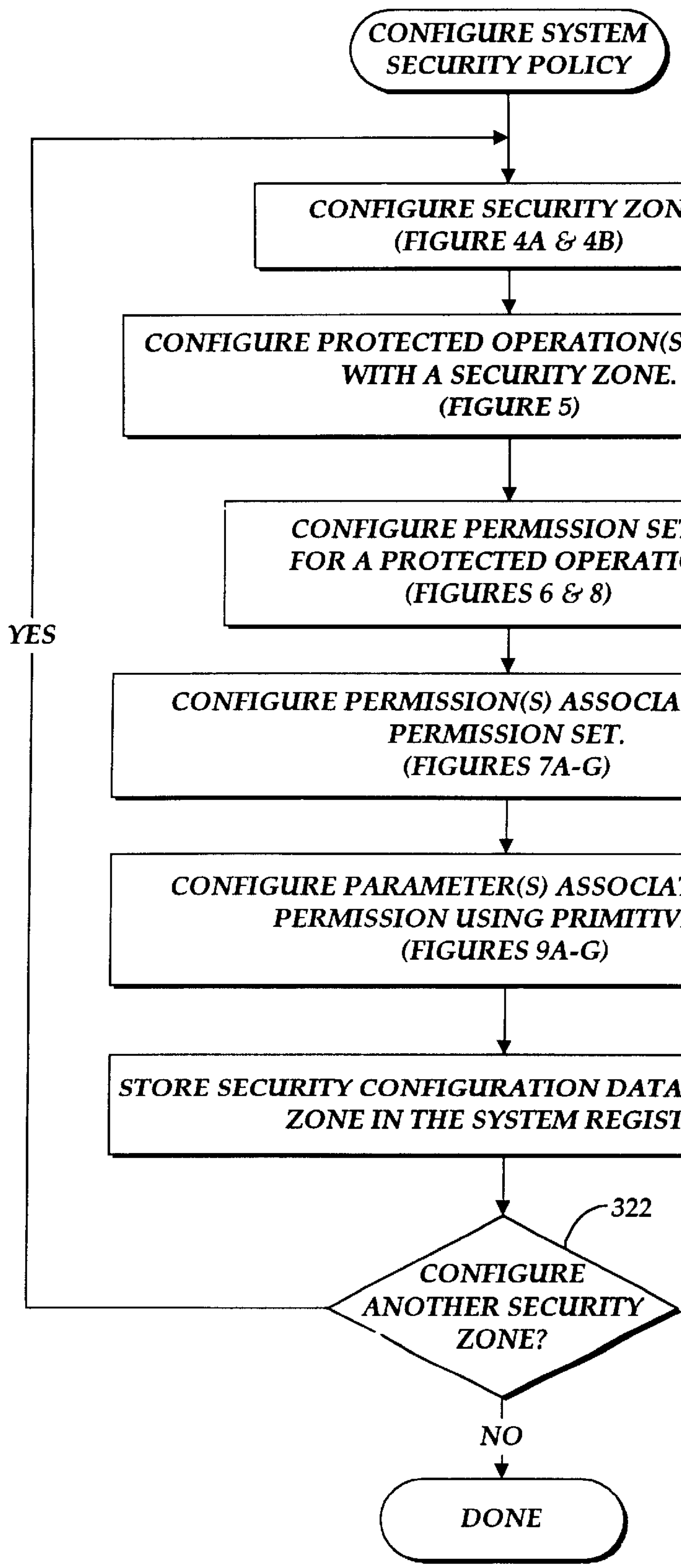

CONFIGURE SYSTEM SECURITY POLICY
310
CONFIGURE SECURITY ZONE. (FIGURE 4A & 4B)
312
CONFIGURE PROTECTED OPERATION(S) ASSOCIATED WITH A SECURITY ZONE. (FIGURE 5)
314
CONFIGURE PERMISSION SET(S) FOR A PROTECTED OPERATION. (FIGURES 6 & 8)
YES
316
CONFIGURE PERMISSION(S) ASSOCIATED WITH A PERMISSION SET. (FIGURES 7A-G)
318
CONFIGURE PARAMETER(S) ASSOCIATED WITH A PERMISSION USING PRIMITIVE(S). (FIGURES 9A-G)
320
STORE SECURITY CONFIGURATION DATA FOR SECURITY ZONE IN THE SYSTEM REGISTRY
322
CONFIGURE ANOTHER SECURITY ZONE?
NO
DONE

Figure 3

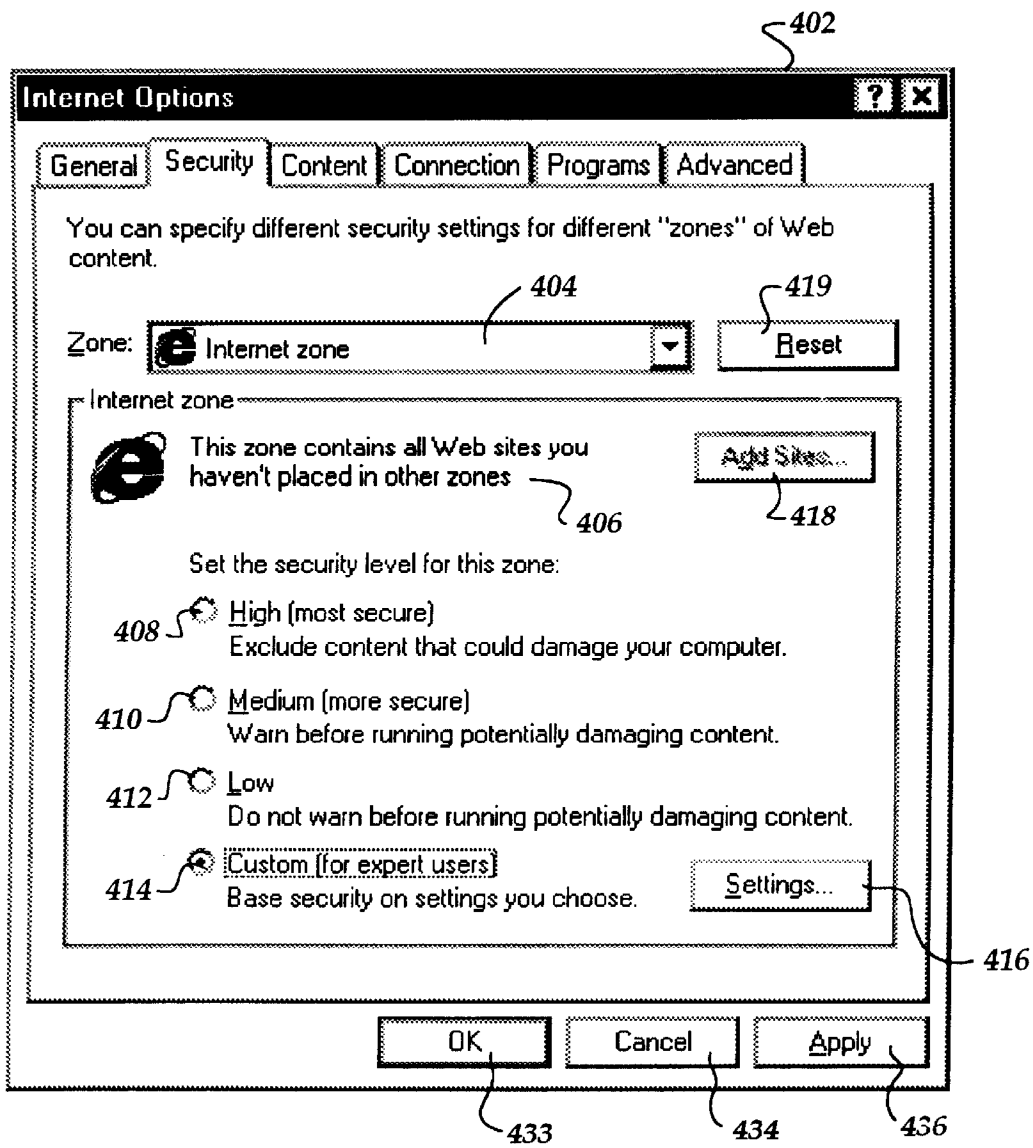

402
Internet Options
General
Security
Content
Connection
Programs
Advanced
You can specify different security settings for different "zones" of Web content.
404
419
Zone:
Internet zone
Reset
Internet zone
This zone contains all Web sites you haven't placed in other zones
Add Sites...
406
418
Set the security level for this zone:
408
High (most secure)
Exclude content that could damage your computer.
410
Medium (more secure)
Warn before running potentially damaging content.
412
Low
Do not warn before running potentially damaging content.
414
Custom (for expert users)
Base security on settings you choose.
Settings...
416
OK
Cancel
Apply
433
434
436

Figure 4A

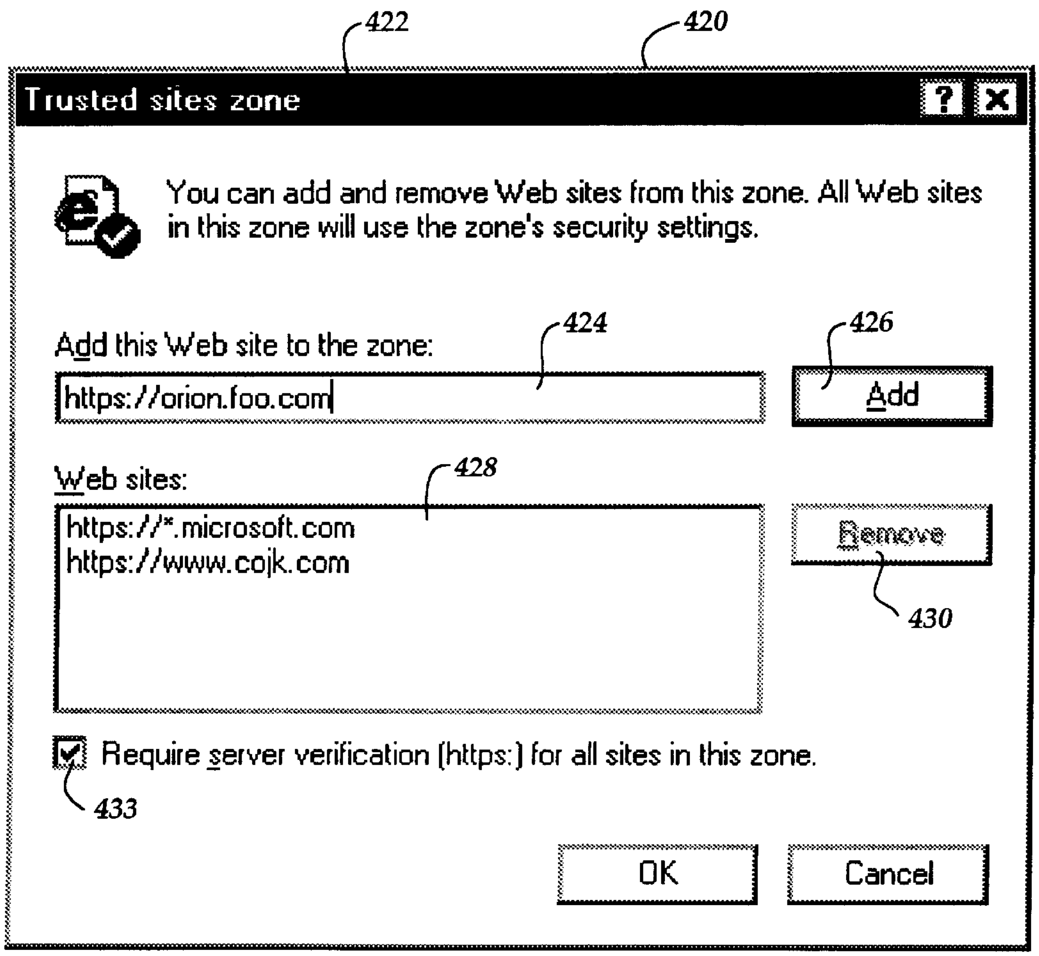
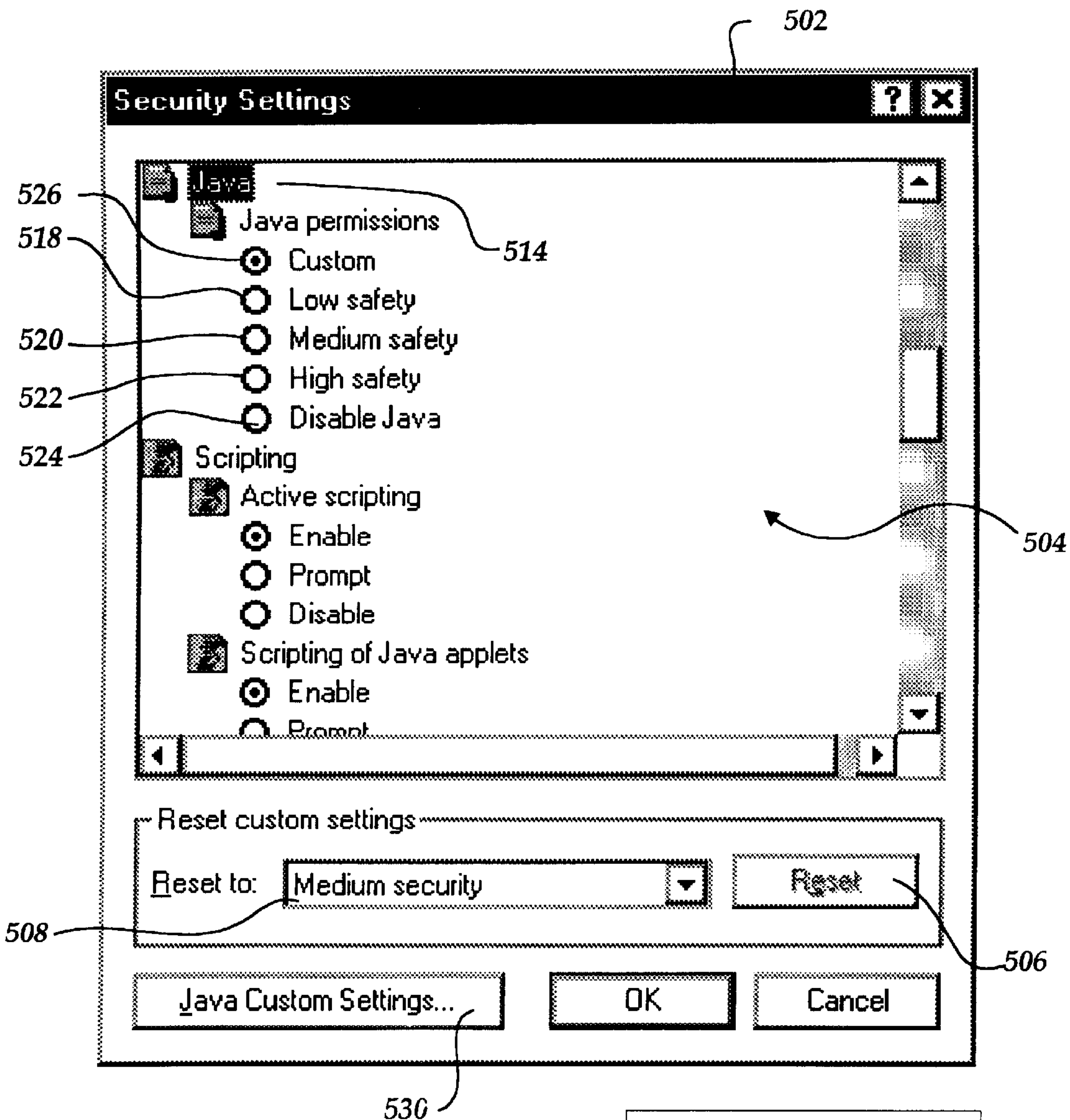

422
420
Trusted sites zone
You can add and remove Web sites from this zone. All Web sites in this zone will use the zone's security settings.
Add this Web site to the zone:
424
426
https://orion.foo.com
Add
Web sites:
428
https://*.microsoft.com
https://www.cojk.com
Remove
430
Require server verification (https:) for all sites in this zone.
433
OK
Cancel

Figure 4B



JAVA Protected Operations

Figure 5A

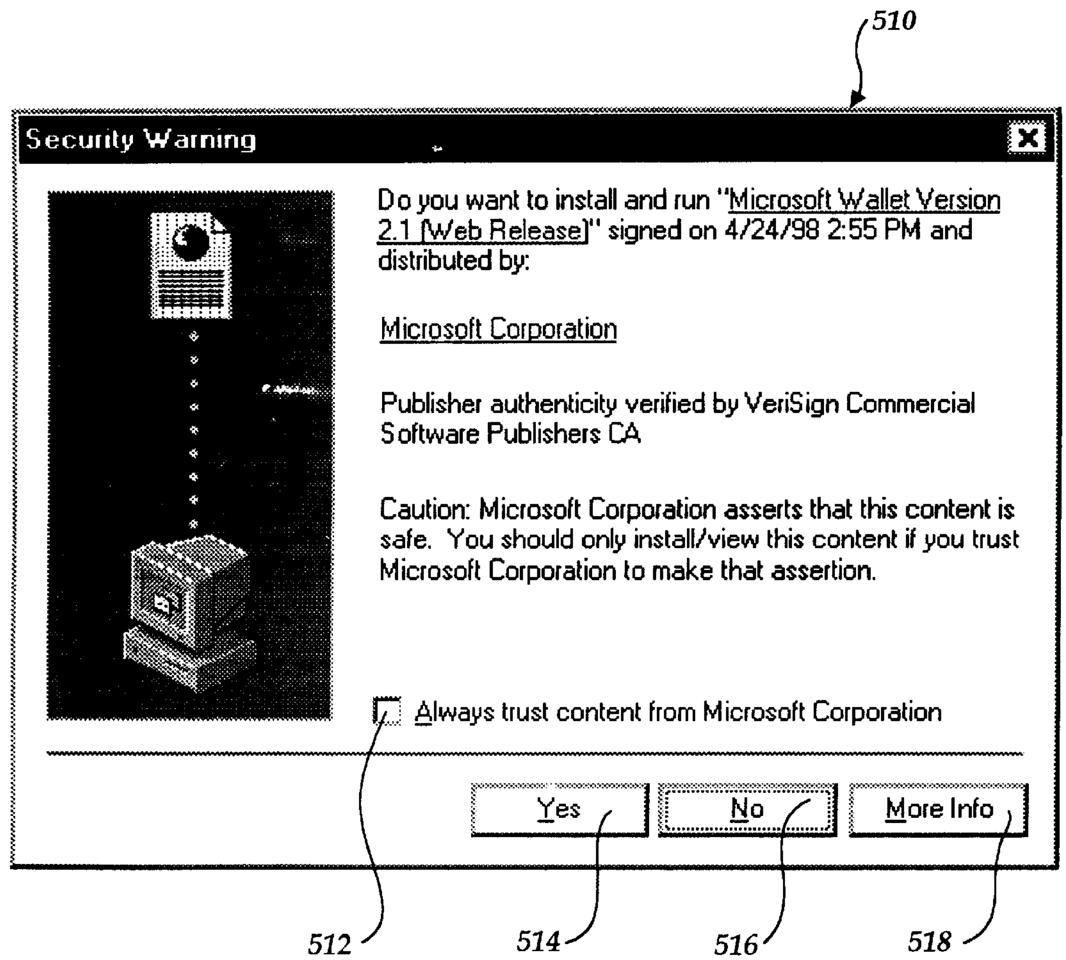

510
Security Warning
Do you want to install and run "Microsoft Wallet Version 2.1 [Web Release]" signed on 4/24/98 2:55 PM and distributed by:
Microsoft Corporation
Publisher authenticity verified by VeriSign Commercial Software Publishers CA
Caution: Microsoft Corporation asserts that this content is safe. You should only install/view this content if you trust Microsoft Corporation to make that assertion.
Always trust content from Microsoft Corporation
Yes
No
More Info
512
514
516
518

Figure 5B

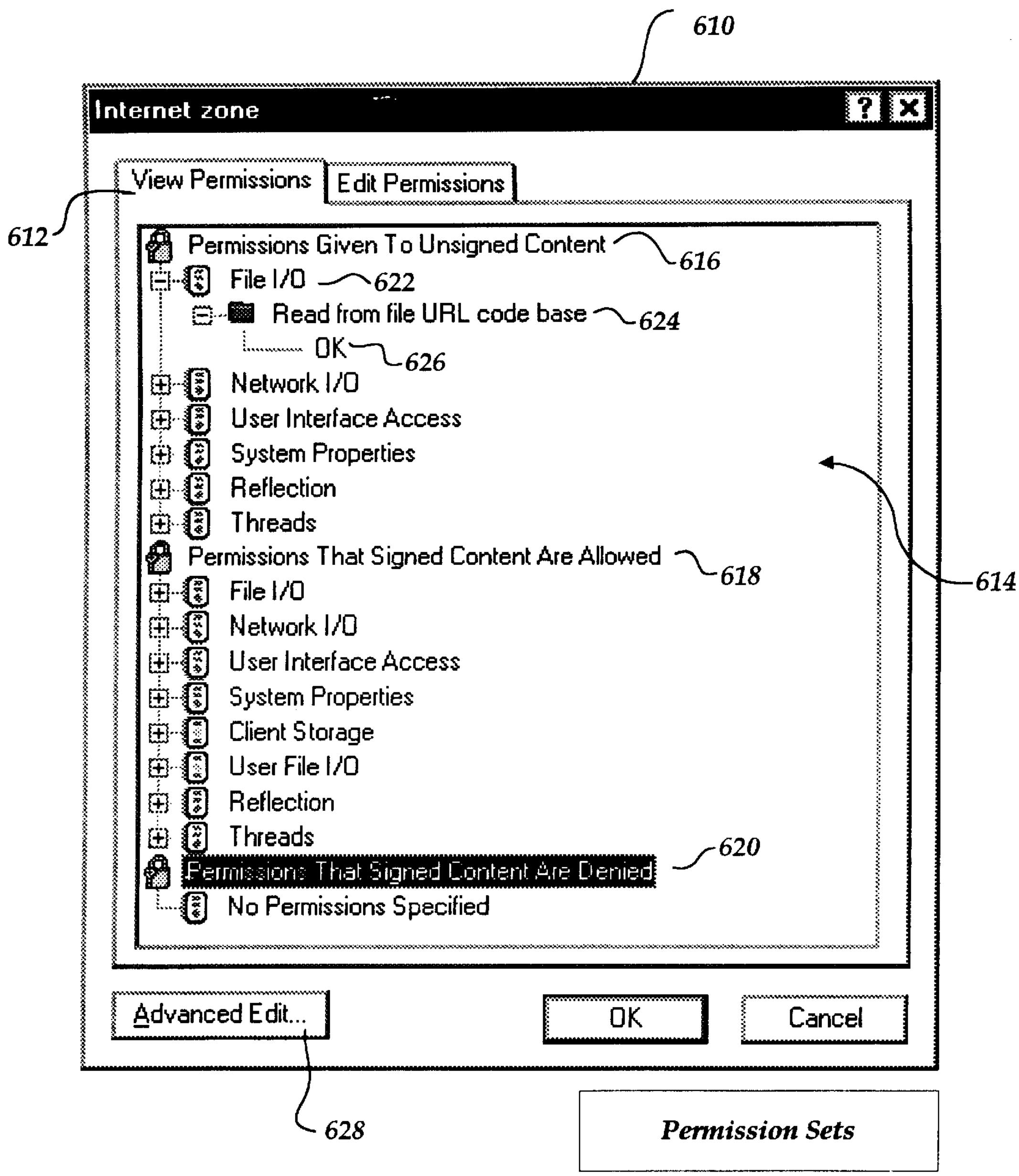

610
Internet zone
View Permissions
Edit Permissions
612
Permissions Given To Unsigned Content
616
File I/O
622
Read from file URL code base
624
OK
626
Network I/O
User Interface Access
System Properties
Reflection
Threads
614
Permissions That Signed Content Are Allowed
618
File I/O
Network I/O
User Interface Access
System Properties
Client Storage
User File I/O
Reflection
Threads
620
Permissions That Signed Content Are Denied
No Permissions Specified
Advanced Edit...
OK
Cancel
628
Permission Sets

Figure 6

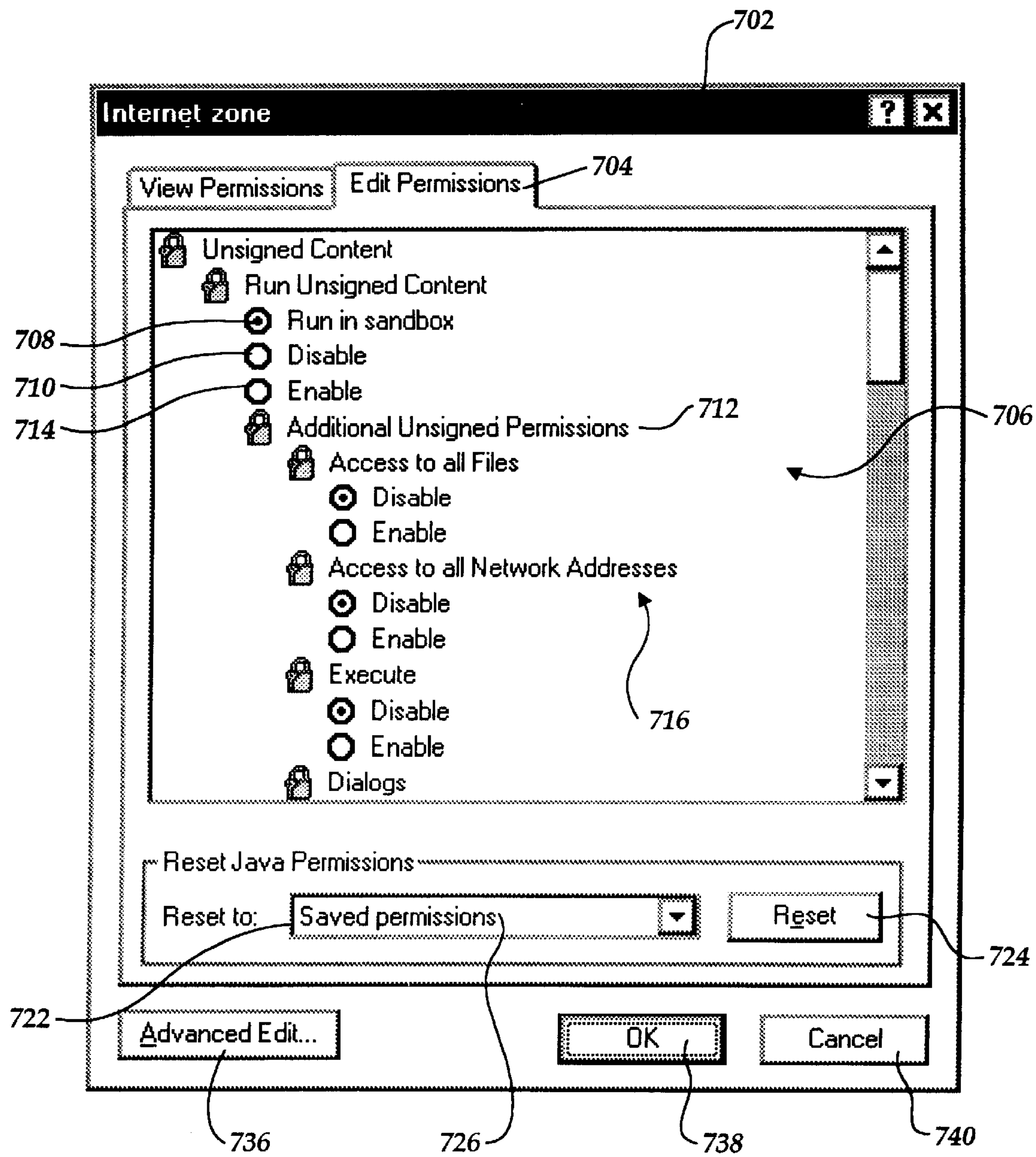

702
Internet zone
View Permissions
Edit Permissions
704
Unsigned Content
Run Unsigned Content
Run in sandbox
Disable
Enable
Additional Unsigned Permissions
712
706
708
710
714
Access to all Files
Disable
Enable
Access to all Network Addresses
Disable
Enable
Execute
Disable
Enable
716
Dialogs
Reset Java Permissions
Reset to:
Saved permissions
Reset
724
722
Advanced Edit...
OK
Cancel
736
726
738
740

Figure 7A

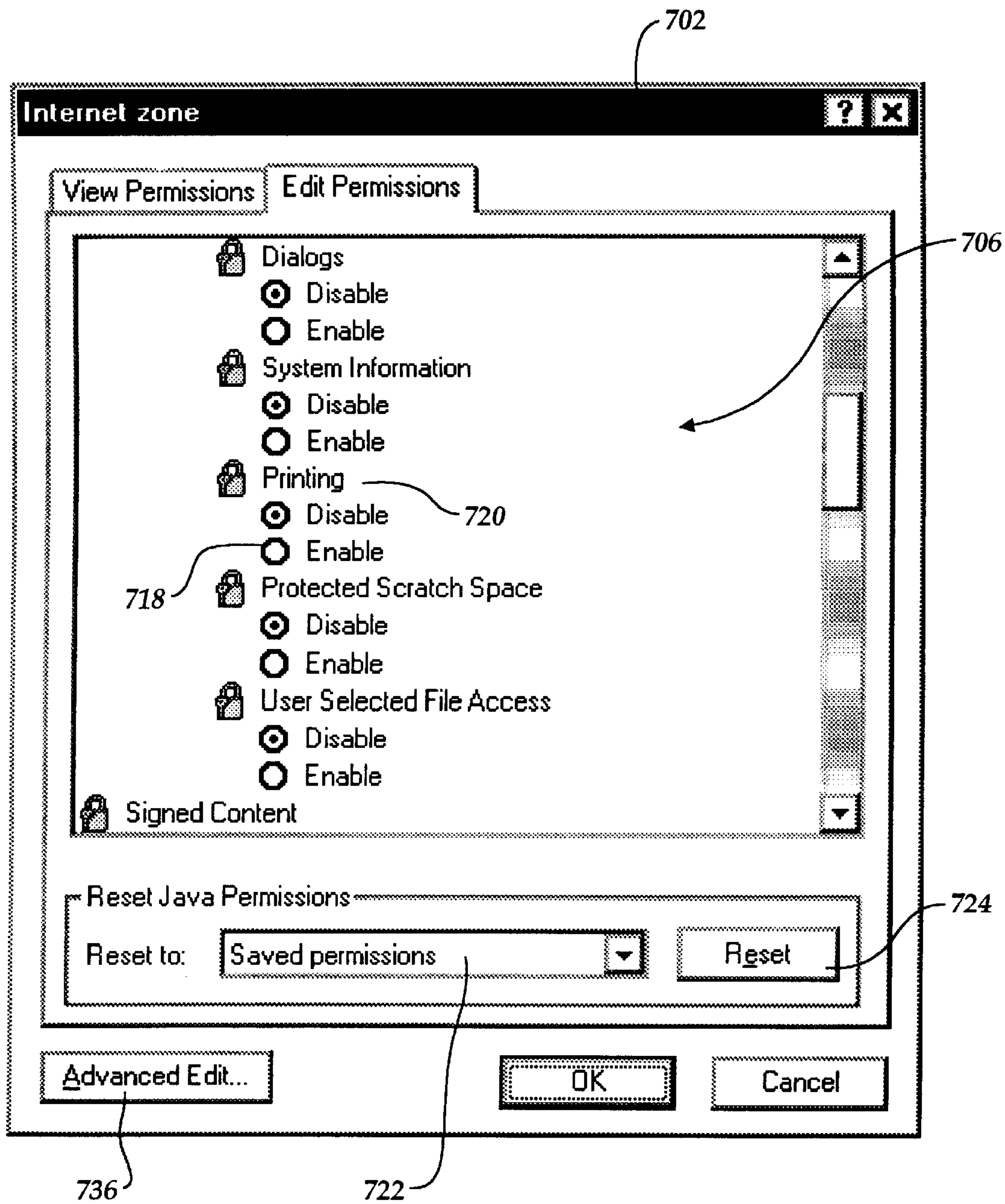

702
Internet zone
View Permissions
Edit Permissions
706
Dialogs
Disable
Enable
System Information
Disable
Enable
Printing
720
Disable
Enable
718
Protected Scratch Space
Disable
Enable
User Selected File Access
Disable
Enable
Signed Content
Reset Java Permissions
724
Reset to:
Saved permissions
Reset
Advanced Edit...
OK
Cancel
736
722

Figure 7B

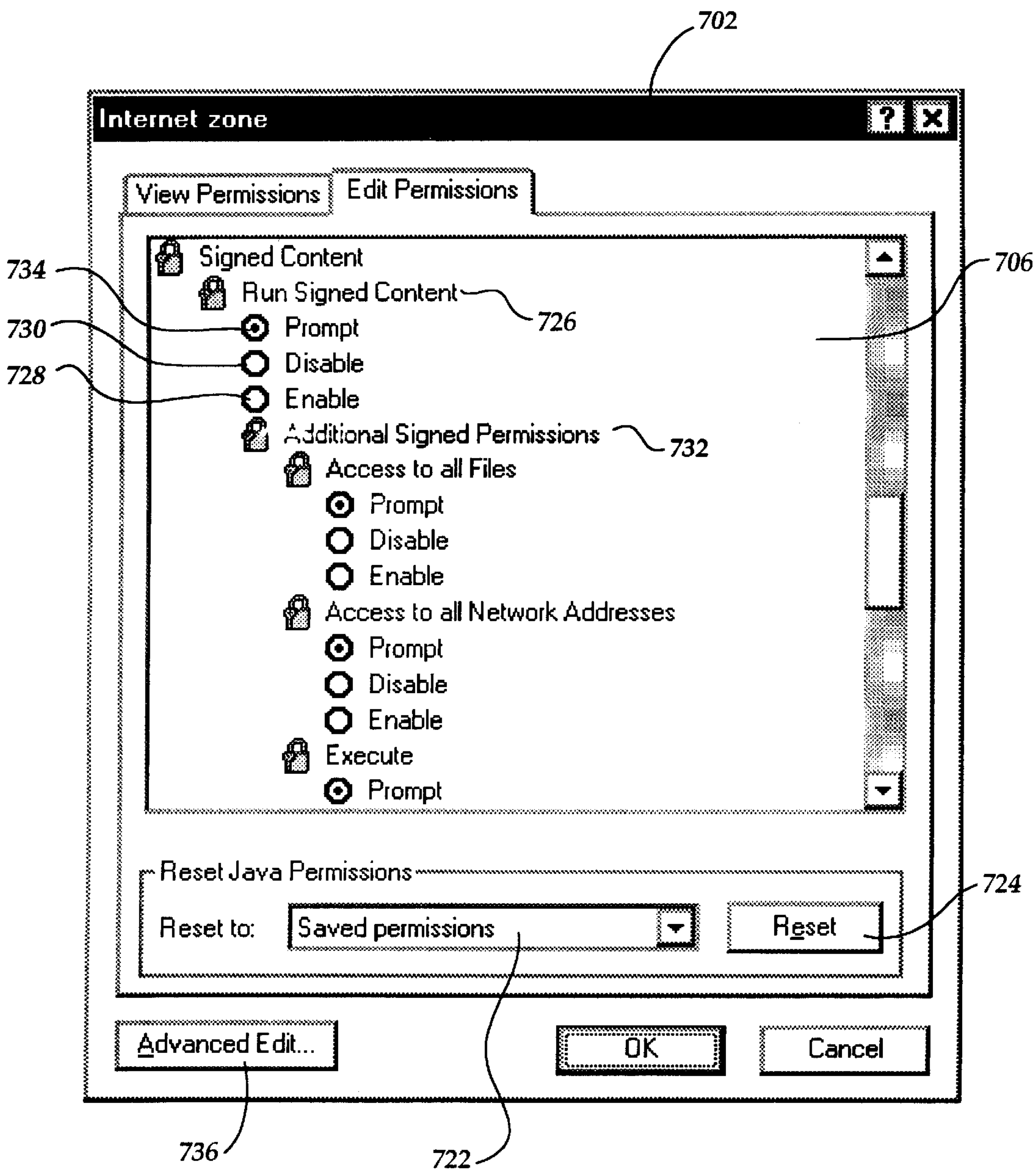

702
Internet zone
View Permissions
Edit Permissions
Signed Content
Run Signed Content
726
Prompt
Disable
Enable
Additional Signed Permissions
732
Access to all Files
Prompt
Disable
Enable
Access to all Network Addresses
Prompt
Disable
Enable
Execute
Prompt
734
730
728
706
Reset Java Permissions
Reset to:
Saved permissions
Reset
724
Advanced Edit...
OK
Cancel
736
722

Figure 7C

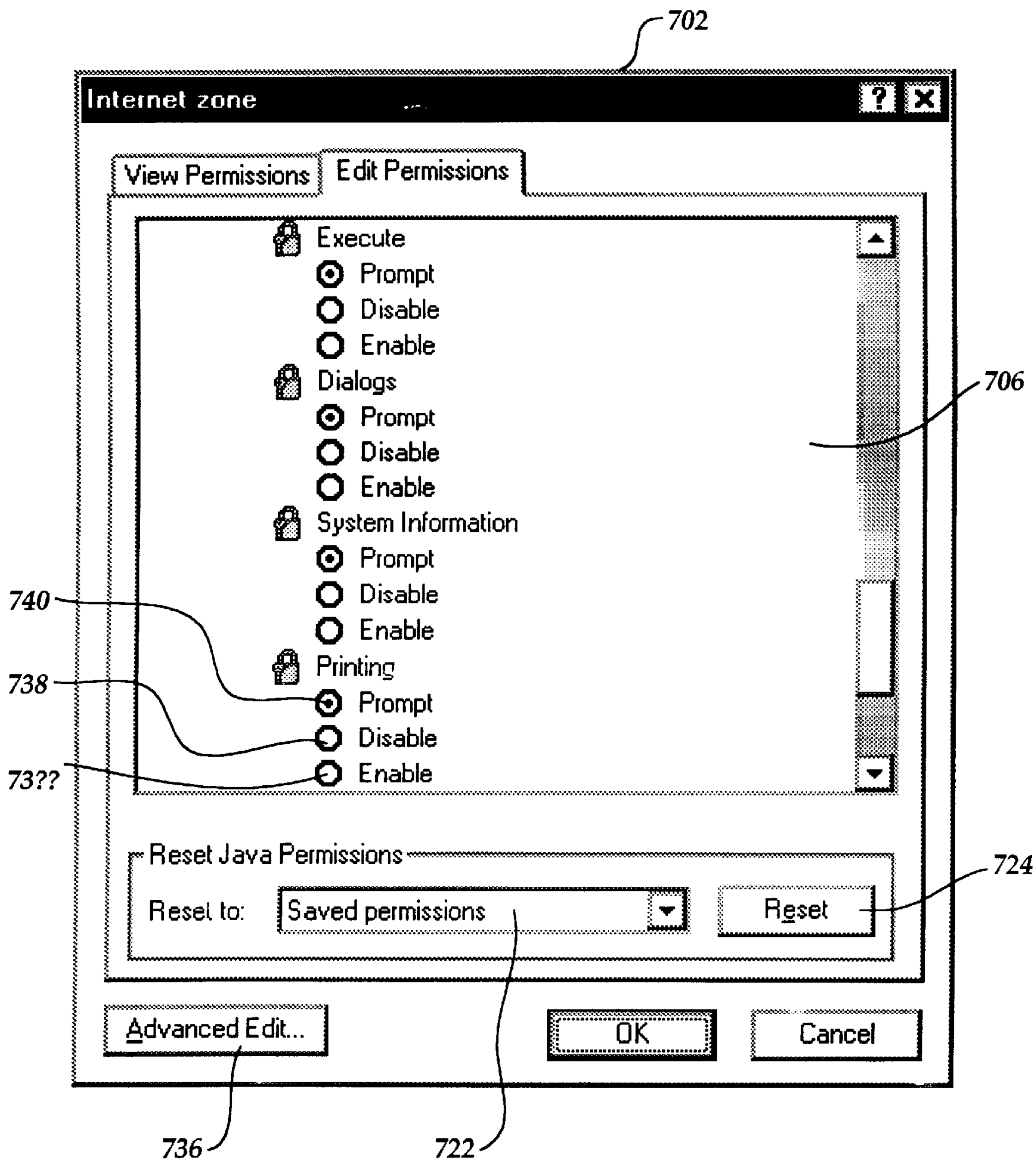

702
Internet zone
View Permissions
Edit Permissions
Execute
Prompt
Disable
Enable
Dialogs
Prompt
Disable
Enable
System Information
Prompt
Disable
Enable
Printing
Prompt
Disable
Enable
706
740
738
73??
Reset Java Permissions
Reset to:
Saved permissions
Reset
724
Advanced Edit...
OK
Cancel
736
722

Figure 7D

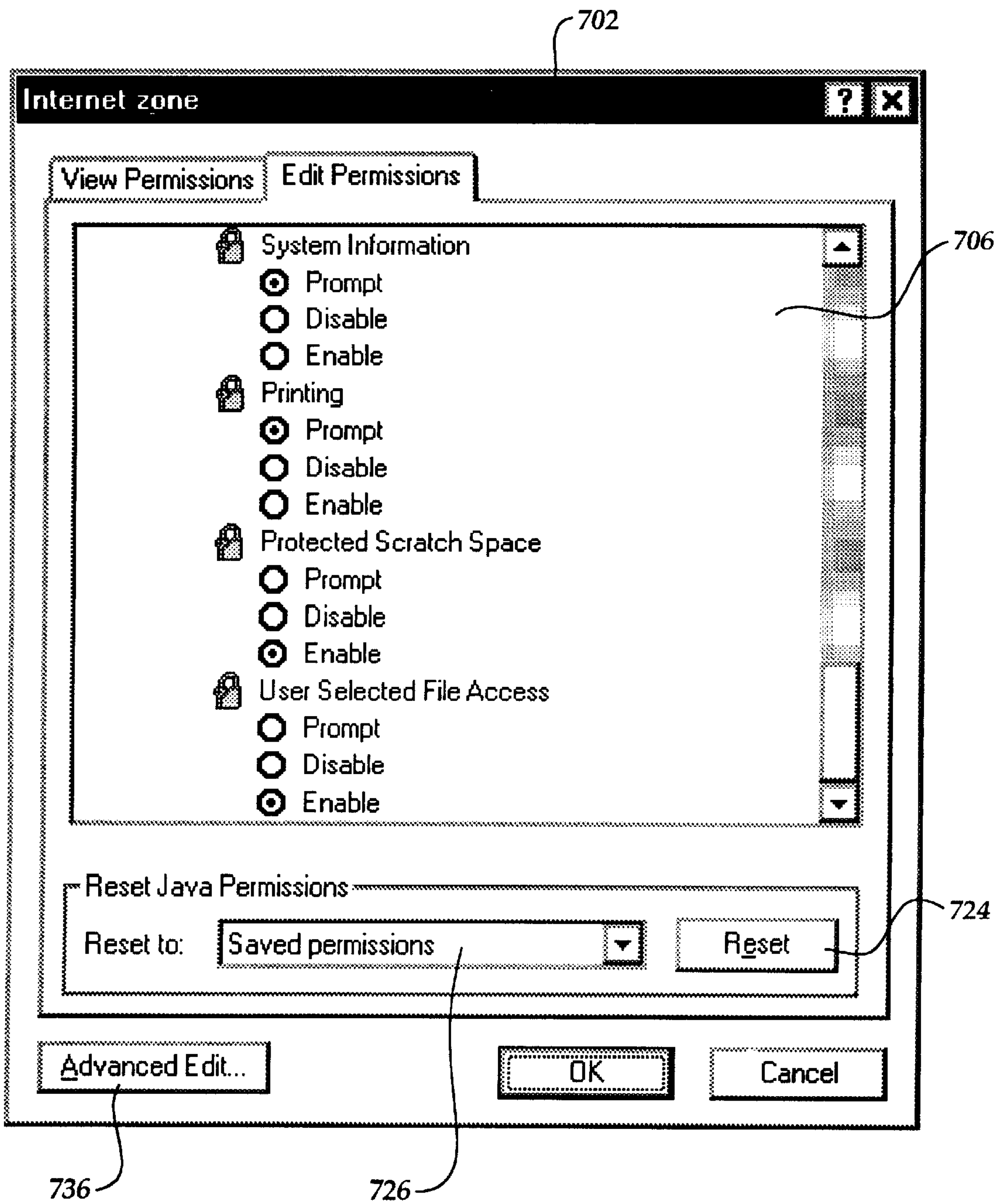

702
Internet zone
View Permissions
Edit Permissions
System Information
Prompt
Disable
Enable
Printing
Prompt
Disable
Enable
Protected Scratch Space
Prompt
Disable
Enable
User Selected File Access
Prompt
Disable
Enable
706
Reset Java Permissions
Reset to:
Saved permissions
Reset
724
Advanced Edit...
OK
Cancel
736
726

Figure 7E

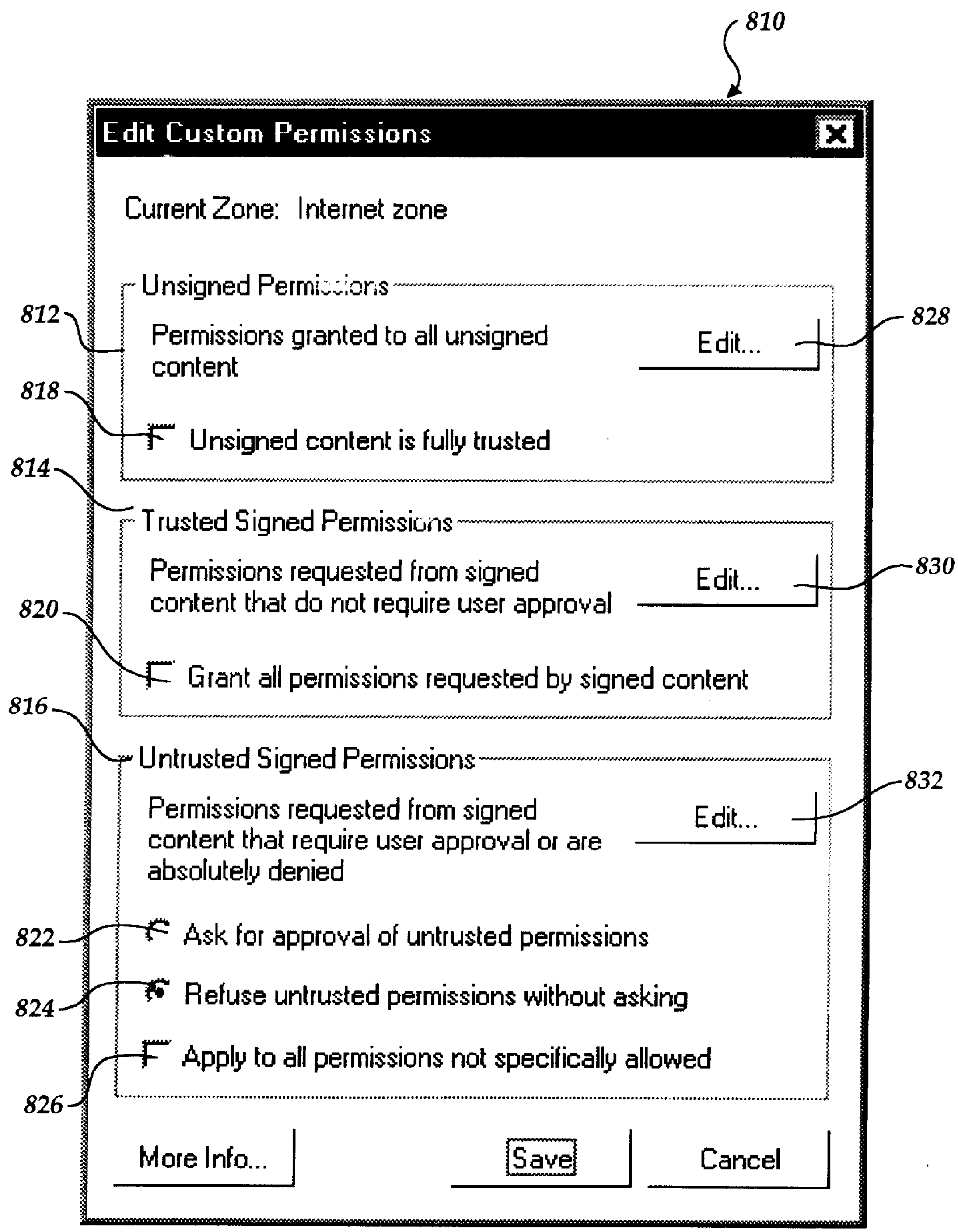

810
Edit Custom Permissions
Current Zone: Internet zone
Unsigned Permissions
812
Permissions granted to all unsigned content
Edit...
828
818
Unsigned content is fully trusted
814
Trusted Signed Permissions
Permissions requested from signed content that do not require user approval
Edit...
830
820
Grant all permissions requested by signed content
816
Untrusted Signed Permissions
Permissions requested from signed content that require user approval or are absolutely denied
Edit...
832
822
Ask for approval of untrusted permissions
824
Refuse untrusted permissions without asking
Apply to all permissions not specifically allowed
826
More Info...
Save
Cancel

Figure 8

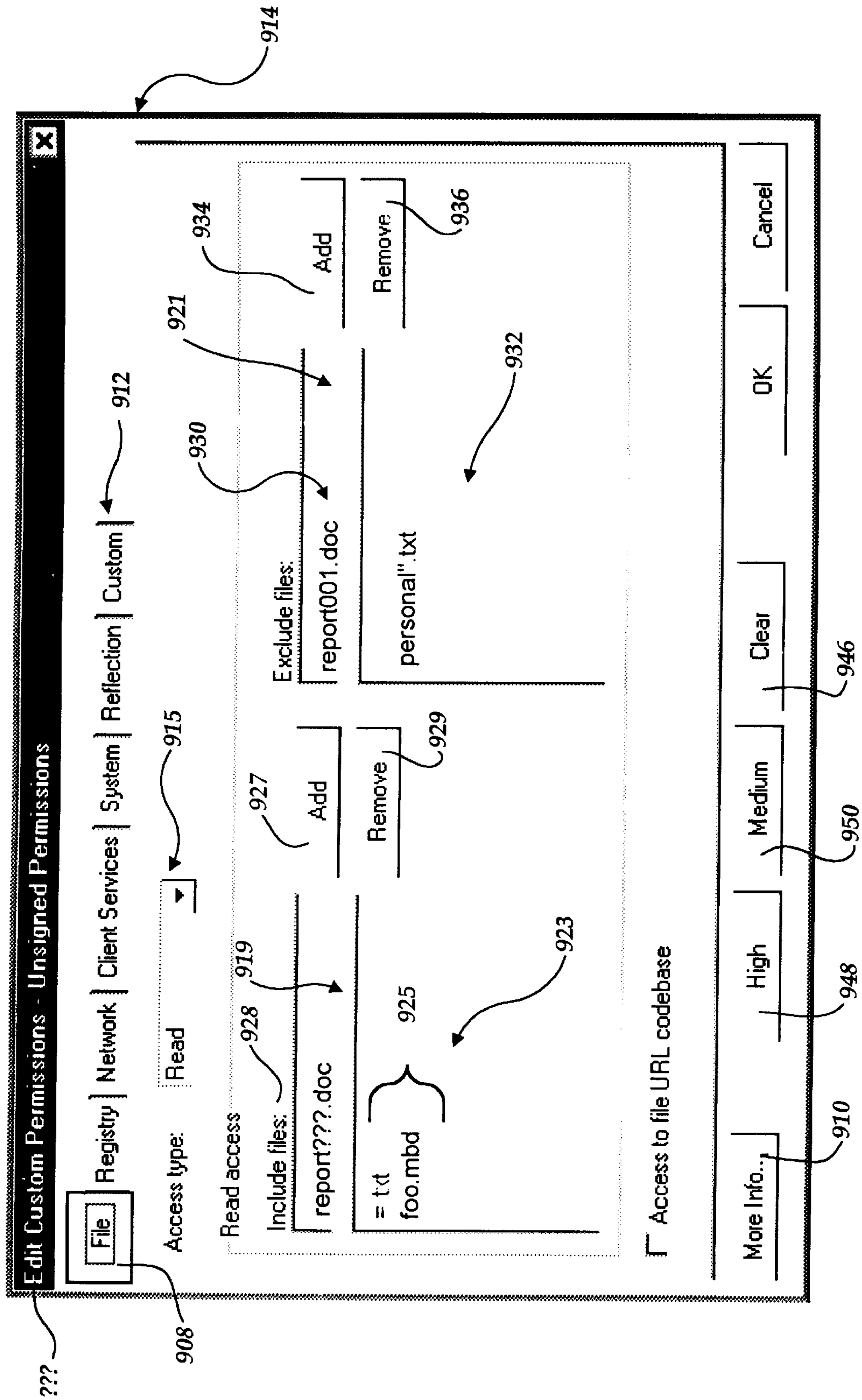

Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
Access type:
Read
Read access
Include files:
report???.doc
= txt
foo.mbd
Add
Remove
Exclude files:
report001.doc
personal".txt
Add
Remove
Access to file URL codebase
More Info...
High
Medium
Clear
OK
Cancel
914
912
915
919
921
923
925
927
928
929
930
932
934
936
908
910
946
948
950
???

Figure 9A

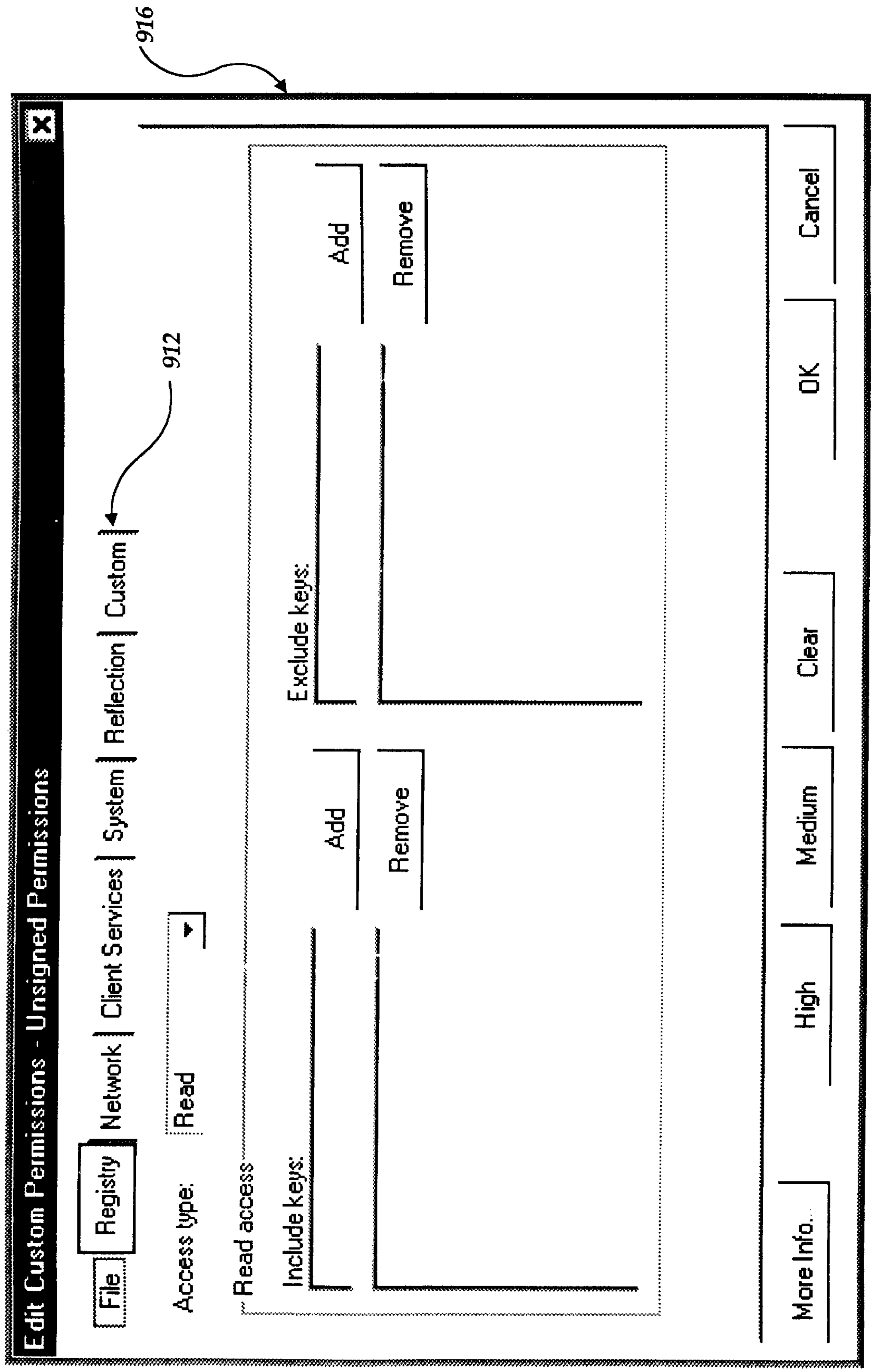

916
Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
912
Access type:
Read
Read access
Include keys:
Add
Remove
Exclude keys:
Add
Remove
More Info...
High
Medium
Clear
OK
Cancel

Figure 9B

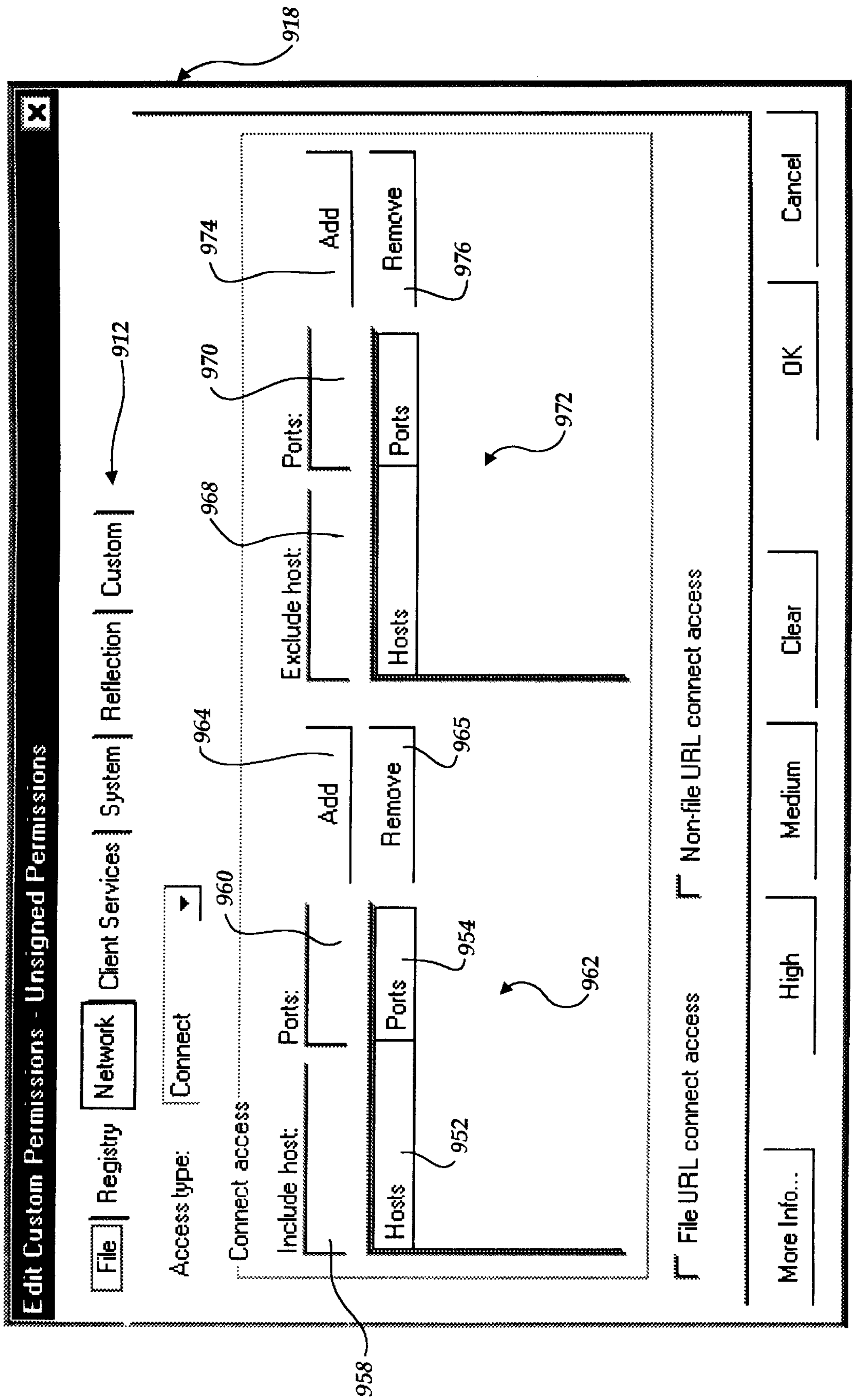

Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
Access type:
Connect
Connect access
Include host:
Ports:
Add
Remove
Hosts
Ports
Exclude host:
Ports:
Add
Remove
Hosts
Ports
File URL connect access
Non-file URL connect access
More Info...
High
Medium
Clear
OK
Cancel
918
912
958
960
962
952
954
964
965
968
970
972
974
976

Figure 9C

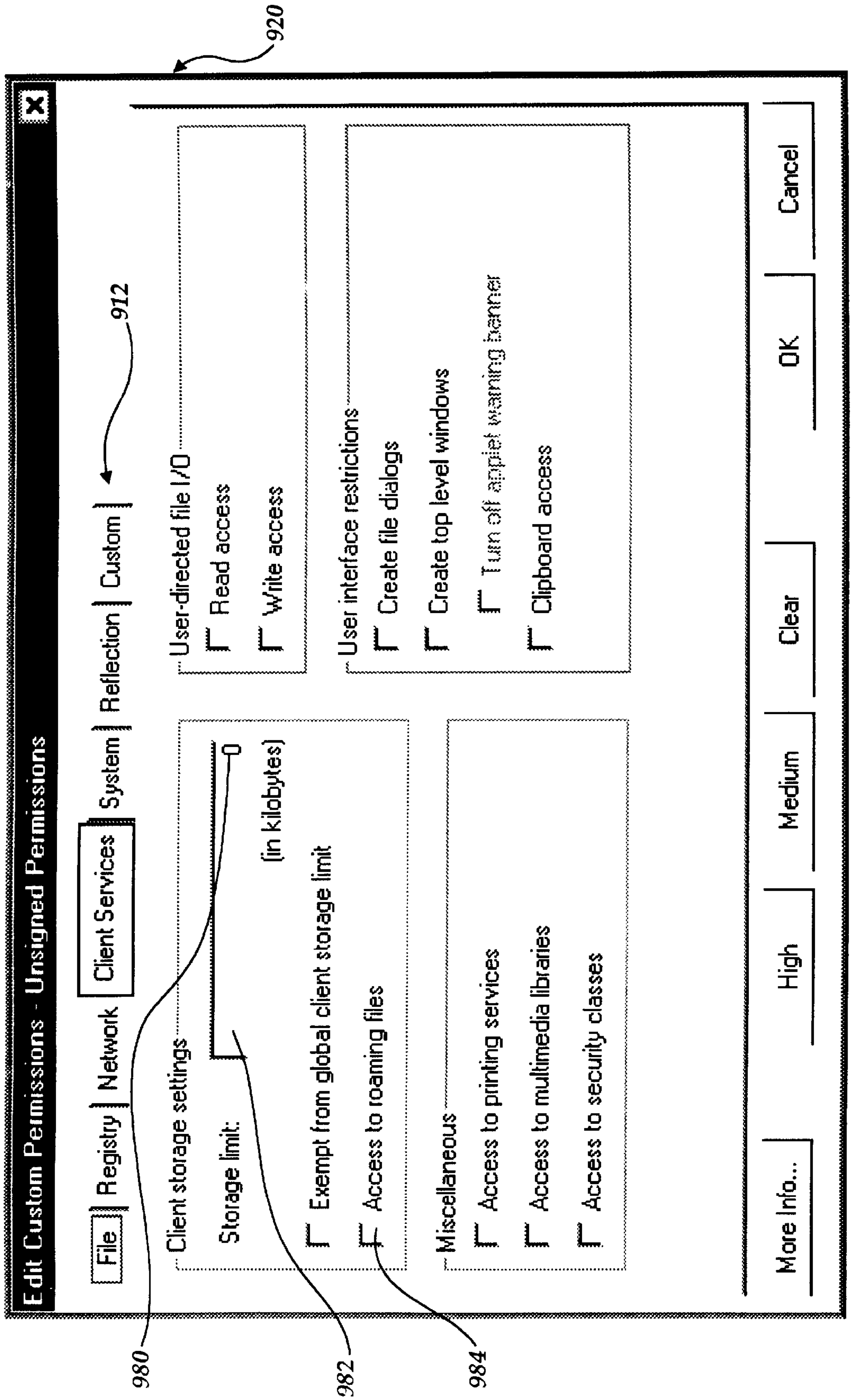

920
Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
912
Client storage settings
Storage limit:
0
(in kilobytes)
Exempt from global client storage limit
Access to roaming files
Miscellaneous
Access to printing services
Access to multimedia libraries
Access to security classes
User-directed file I/O
Read access
Write access
User interface restrictions
Create file dialogs
Create top level windows
Turn off applet warning banner
Clipboard access
More Info...
High
Medium
Clear
OK
Cancel
980
982
984

Figure 9D

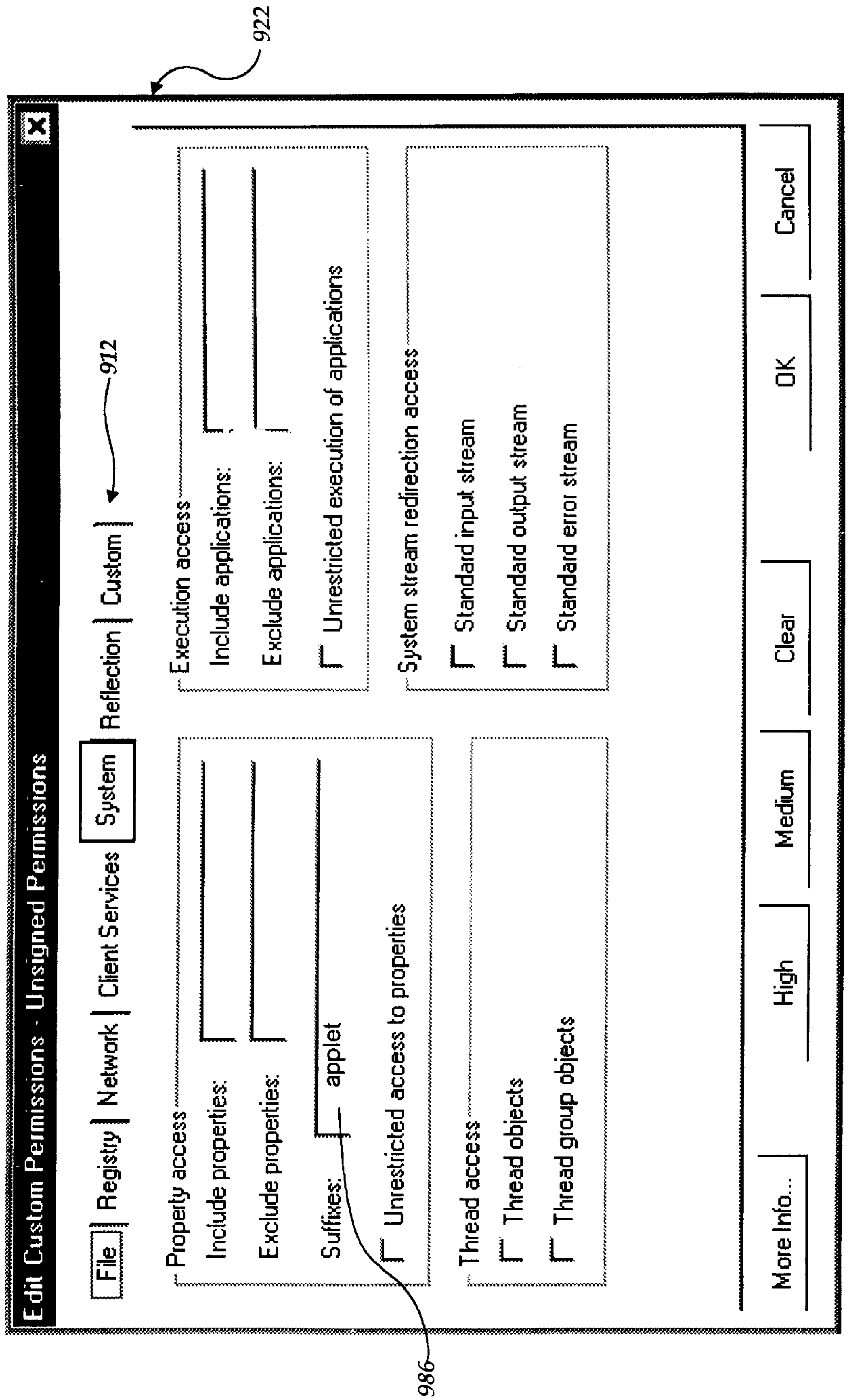

922
Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
912
Property access
Include properties:
Exclude properties:
Suffixes:
applet
Unrestricted access to properties
Thread access
Thread objects
Thread group objects
Execution access
Include applications:
Exclude applications:
Unrestricted execution of applications
System stream redirection access
Standard input stream
Standard output stream
Standard error stream
More Info...
High
Medium
Clear
OK
Cancel
986

Figure 9E

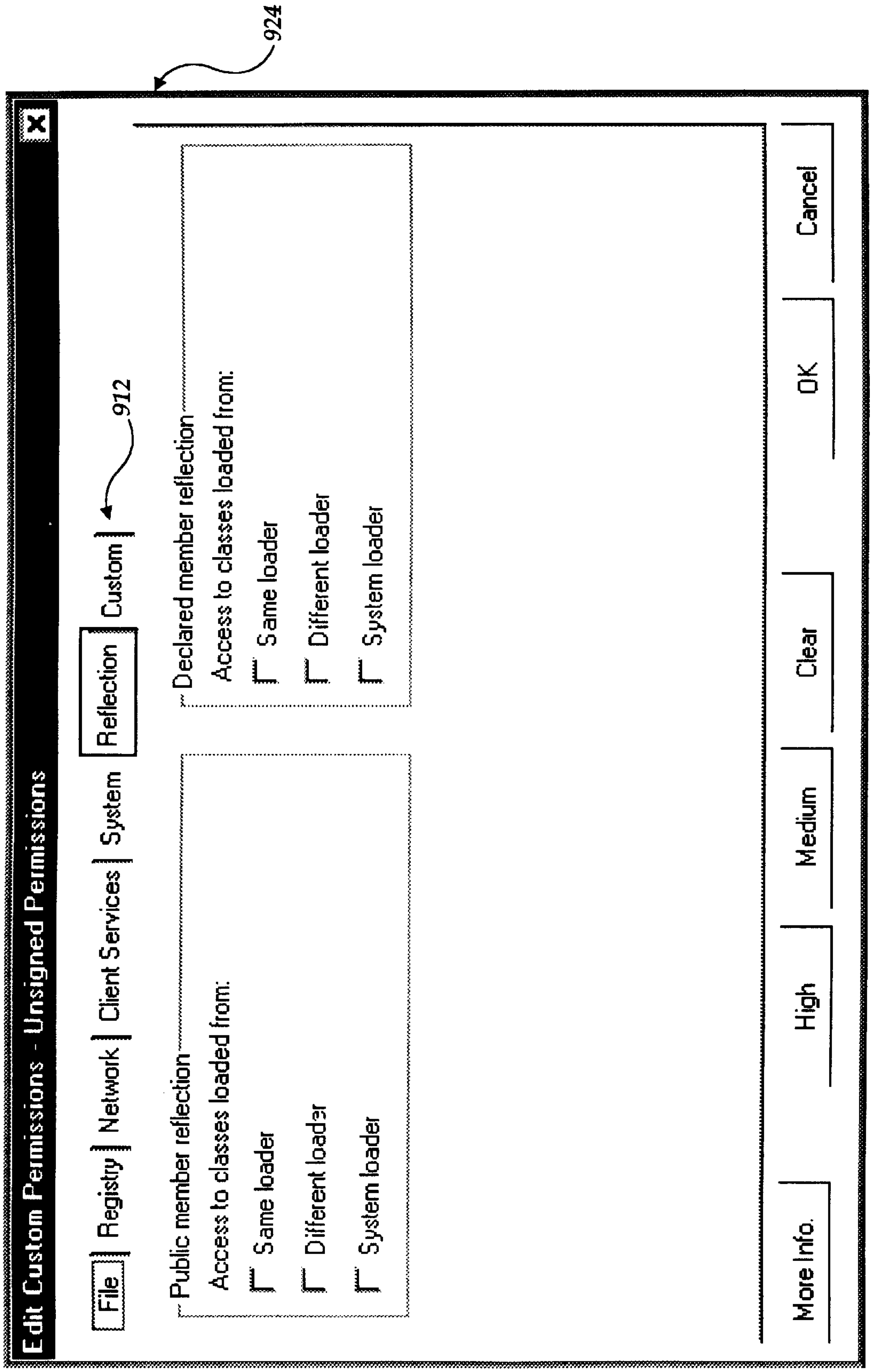

924
Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
912
Public member reflection
Access to classes loaded from:
Same loader
Different loader
System loader
Declared member reflection
Access to classes loaded from:
Same loader
Different loader
System loader
More Info.
High
Medium
Clear
OK
Cancel

Figure 9F

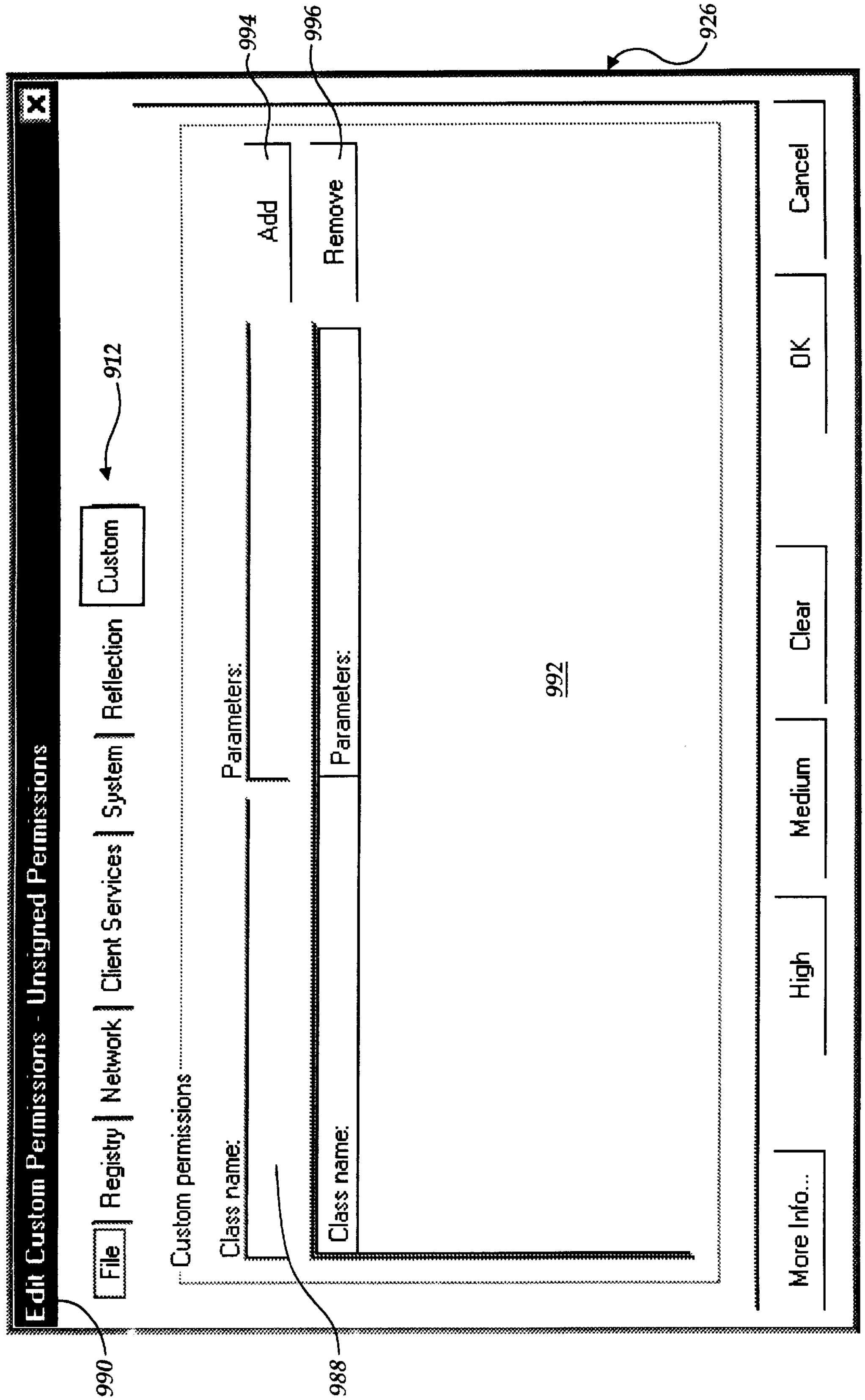

Edit Custom Permissions - Unsigned Permissions
File
Registry
Network
Client Services
System
Reflection
Custom
912
Custom permissions
Class name:
Parameters:
Add
Remove
Class name:
Parameters:
992
More Info...
High
Medium
Clear
OK
Cancel
926
994
996
990
988

Figure 9G

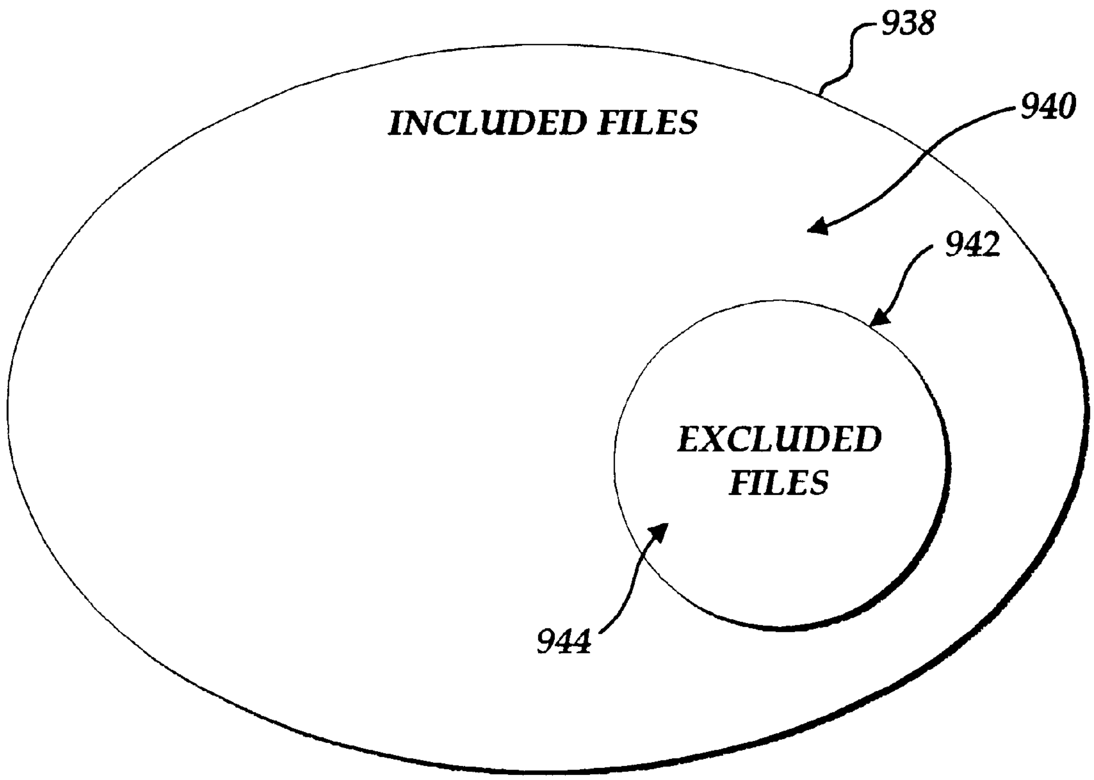


Figure 9H

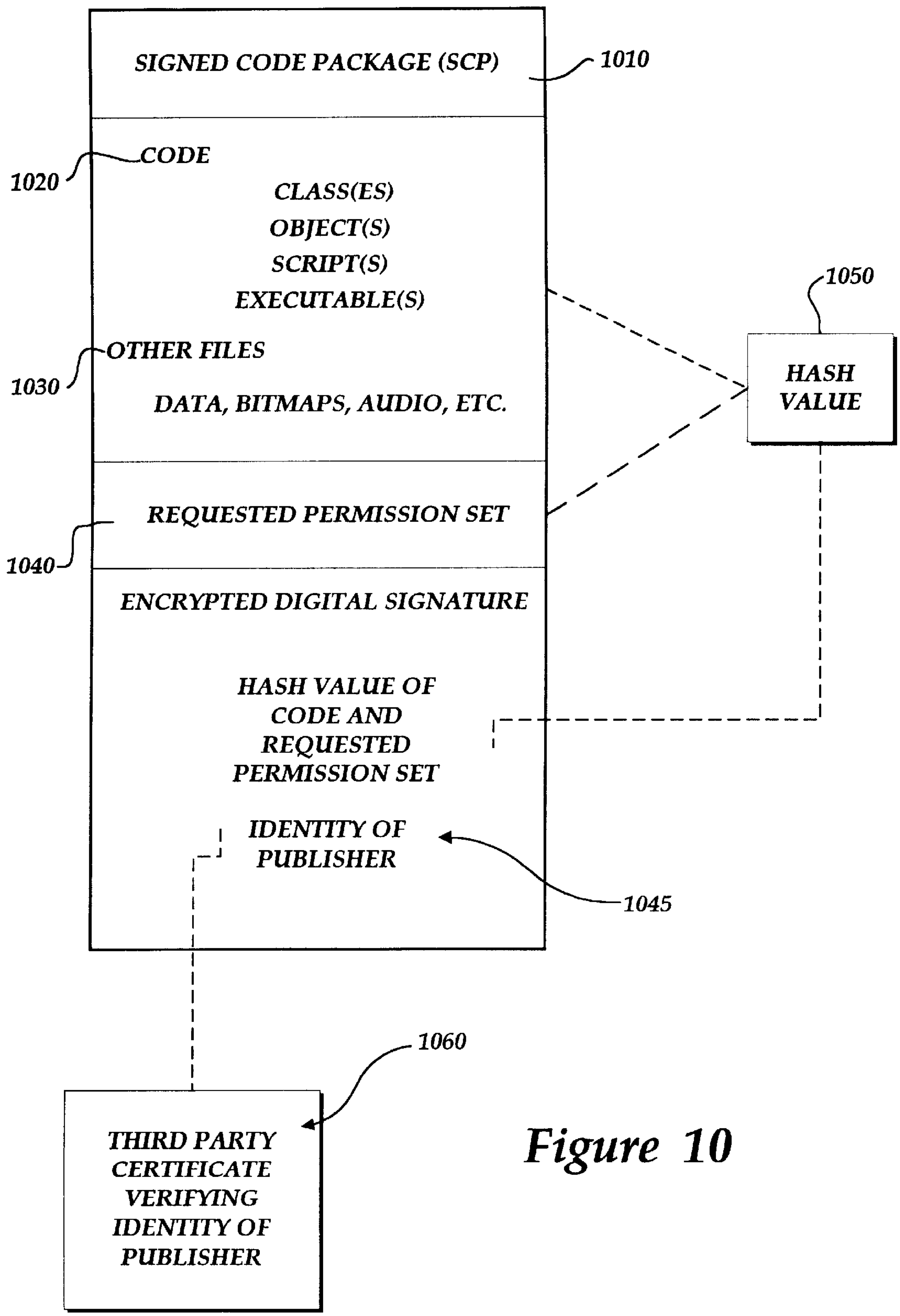

SIGNED CODE PACKAGE (SCP)
1010
1020
CODE
CLASS(ES)
OBJECT(S)
SCRIPT(S)
EXECUTABLE(S)
1030
OTHER FILES
DATA, BITMAPS, AUDIO, ETC.
1040
REQUESTED PERMISSION SET
ENCRYPTED DIGITAL SIGNATURE
HASH VALUE OF CODE AND REQUESTED PERMISSION SET
IDENTITY OF PUBLISHER
1045
1050
HASH VALUE
1060
THIRD PARTY CERTIFICATE VERIFYING IDENTITY OF PUBLISHER

Figure 10

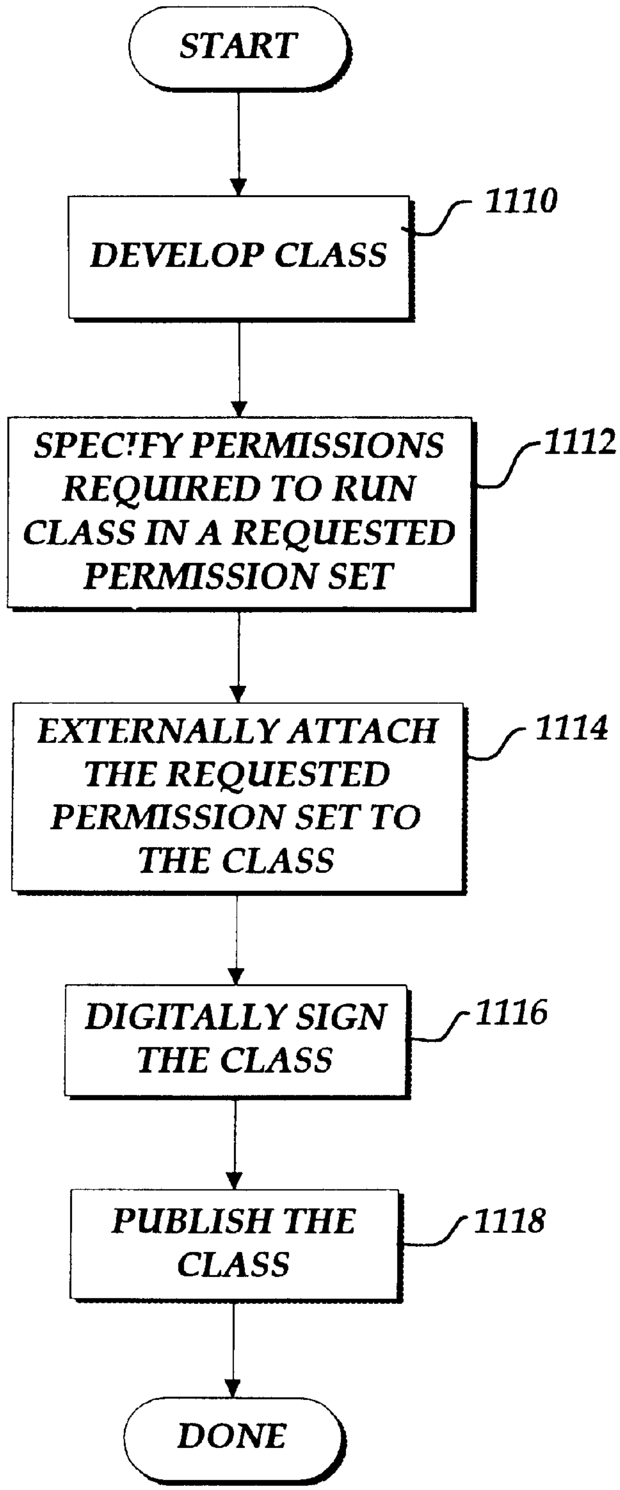

START
DEVELOP CLASS
1110
SPECIFY PERMISSIONS REQUIRED TO RUN CLASS IN A REQUESTED PERMISSION SET
1112
EXTERNALLY ATTACH THE REQUESTED PERMISSION SET TO THE CLASS
1114
DIGITALLY SIGN THE CLASS
1116
PUBLISH THE CLASS
1118
DONE

Figure 11

1202

```
; PermissionDataSet INI encoding
;
; Sample INI File
;
;

; A semi-colon (;) at the beginning of a line signifies a
; comment to the end of the current line.

; AllowActiveX tells the sign tool to sign with full ActiveX
; permissions, in addition to any other permissions.
;[AllowActiveX]

; If the FullyTrusted section exists, all other
; permissions are ignored.
;[FullyTrusted]

;
; ExecutionPermission
;
[com.ms.security.permissions.ExecutionPermission]
```

1230a

```
; If Unrestricted is true, the other settings are ignored.
;Unrestricted=true
Unrestricted=false
IncludeNames=*.exe;*.txt;activextest
ExcludeNames=foo.exe
```

1232a

```
;
; ClientStoragePermission
;
[com.ms.security.permissions.ClientStoragePermission]
```

1230b

```
; Limit is in bytes.
Limit=100
RoamingFiles=true
GlobalExempt=true
```

1232b

```
;
;
```

Figure 12A

```
; FileIOPermission
;                      1216
[com.ms.security.permissions.FileIOPermission]   1230c
IncludeRead=*.txt;foo.mdb   1214
ExcludeRead=crayon
IncludeWrite=89kandaf;-l?daf;s
ExcludeWrite=s;-lddaf                              1232c
IncludeDelete=45*.4;*.*.*.*;****a;??4nf
ExcludeDelete=ff4nf
ReadFileURLCodebase=true

;
; MultimediaPermission
;                                                   1230d
[com.ms.security.permissions.MultimediaPermission]
; No specific settings required for MultimediaPermission   1232d

;
; NetIOPermission
;
[com.ms.security.permissions.NetIOPermission]      1230e
IncludeConnectIPs=1.2.3.4:1-102
ExcludeConnectIPs=
IncludeBindIPs=4.4.4.4:4-55,97;4.4.4.4:4
ExcludeBindIPs=4.4.4.4:8
IncludeMulticastIPs=
ExcludeMulticastIPs=
IncludeConnectHosts=www.microsoft.com:80
;IncludeConnectHosts=www.microsoft.com
ExcludeConnectHosts=
IncludeBindHosts=                                  1232e
ExcludeBindHosts=
IncludeMulticastHosts=
ExcludeMulticastHosts=
IncludeConnectGlobalPorts=3-100, 999
ExcludeConnectGlobalPorts=43, 999
IncludeBindGlobalPorts=43
ExcludeBindGlobalPorts=
ConnectToFileURLCodebase=true
ConnectToNonFileURLCodebase=true
```

Figure 12B

```
; PrintingPermission
;
[com.ms.security.permissions.PrintingPermission]    1230f
; No specific settings required for PrintingPermission    1232f

;
; PropertyPermission
;
[com.ms.security.permissions.PropertyPermission]    1230g
; If Unrestricted is true, the other settings
; are ignored.
Unrestricted=false
AllowedSuffixes=applet;foobar    1232g
IncludedProperties=*system*
ExcludedProperties=*cryptsystem*

;
; ReflectionPermission
;
[com.ms.security.permissions.ReflectionPermission]    1230h
PublicSame=true
PublicDifferent=true
PublicSystem=true
DeclaredSame=true    1232h
DeclaredDifferent=true
DeclaredSystem=true

;
; RegistryPermission
;
[com.ms.security.permissions.RegistryPermission]    1230i
IncludeOpen=hk*
ExcludeOpen=*N
IncludeRead=fdjl
ExcludeRead=
IncludeWrite=*?5$$
ExcludeWrite=gd5$$    1232i
IncludeDelete=*
ExcludeDelete=d
IncludeCreate=*
ExcludeCreate=s
```

Figure 12C

```
; SecurityPermission
;
[com.ms.security.permissions.SecurityPermission]          1230j
; No specific settings required for SecurityPermission    1232j

;
; SystemStreamsPermission
;
[com.ms.security.permissions.SystemStreamsPermission]     1230k
SetSysIn=true
SetSysOut=true                                            1232k
SetSysErr=true

;
; ThreadPermission
;
[com.ms.security.permissions.ThreadPermission]            1230l
AllThreadGroups=true
AllThreads=true                                           1232l

;
; UIPermission
;
[com.ms.security.permissions.UIPermission]                1230m
ClipboardAccess=true
TopLevelWindows=true
NoWarningBanners=false                                    1232m
FileDialogs=true
EventQueueAccess=true

;
; UserFileIOPermission
;
[com.ms.security.permissions.UserFileIOPermission]        1230n
CanRead=true
CanWrite=true                                             1232n
```


Figure 12D

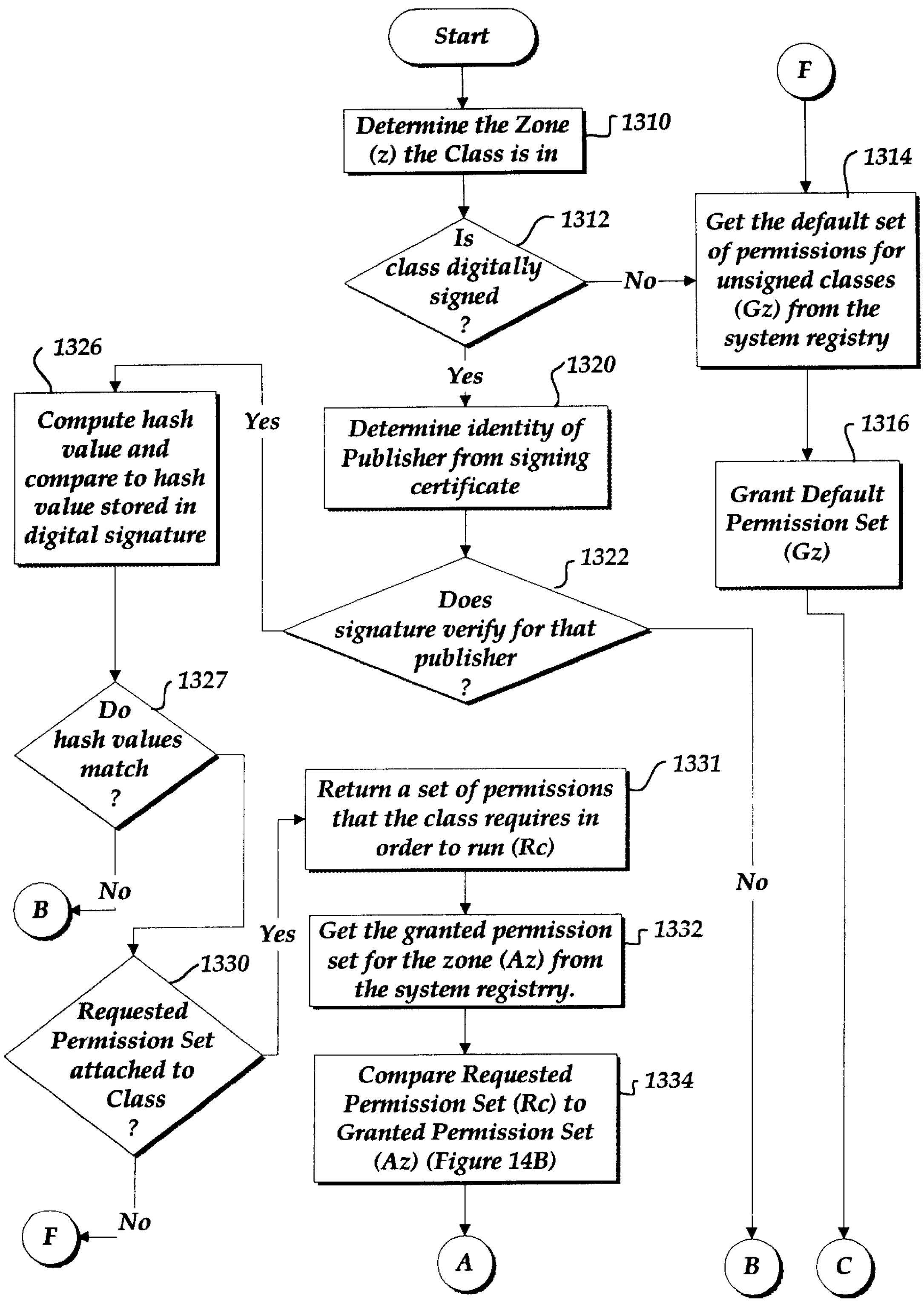

Start
F
Determine the Zone (z) the Class is in
1310
1314
Is class digitally signed ?
1312
No
Get the default set of permissions for unsigned classes (Gz) from the system registry
1326
Yes
1320
Compute hash value and compare to hash value stored in digital signature
Yes
Determine identity of Publisher from signing certificate
1316
Grant Default Permission Set (Gz)
1322
Does signature verify for that publisher ?
1327
Do hash values match ?
1331
Return a set of permissions that the class requires in order to run (Rc)
No
B
No
Yes
Get the granted permission set for the zone (Az) from the system registry.
1332
1330
Requested Permission Set attached to Class ?
Compare Requested Permission Set (Rc) to Granted Permission Set (Az) (Figure 14B)
1334
F
No
A
B
C

Figure 13A

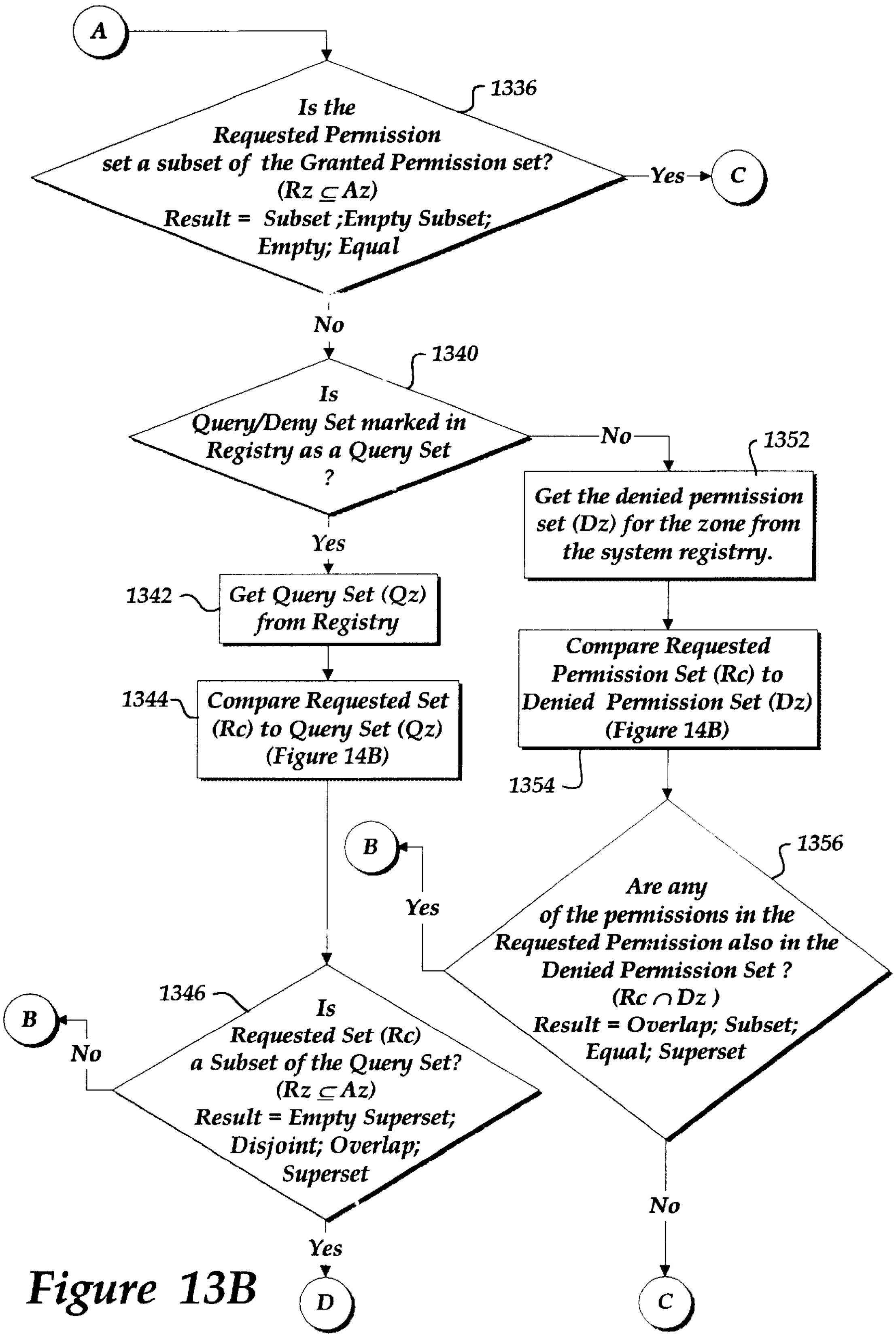

A
Is the Requested Permission set a subset of the Granted Permission set? (Rz ⊆ Az) Result = Subset ;Empty Subset; Empty; Equal
1336
Yes
C
No
Is Query/Deny Set marked in Registry as a Query Set ?
1340
No
Get the denied permission set (Dz) for the zone from the system registrry.
1352
Yes
1342
Get Query Set (Qz) from Registry
1344
Compare Requested Set (Rc) to Query Set (Qz) (Figure 14B)
Compare Requested Permission Set (Rc) to Denied Permission Set (Dz) (Figure 14B)
1354
B
Yes
Are any of the permissions in the Requested Permission also in the Denied Permission Set ? (Rc ∩ Dz) Result = Overlap; Subset; Equal; Superset
1356
1346
B
No
Is Requested Set (Rc) a Subset of the Query Set? (Rz ⊆ Az) Result = Empty Superset; Disjoint; Overlap; Superset
Yes
D
No
C

Figure 13B

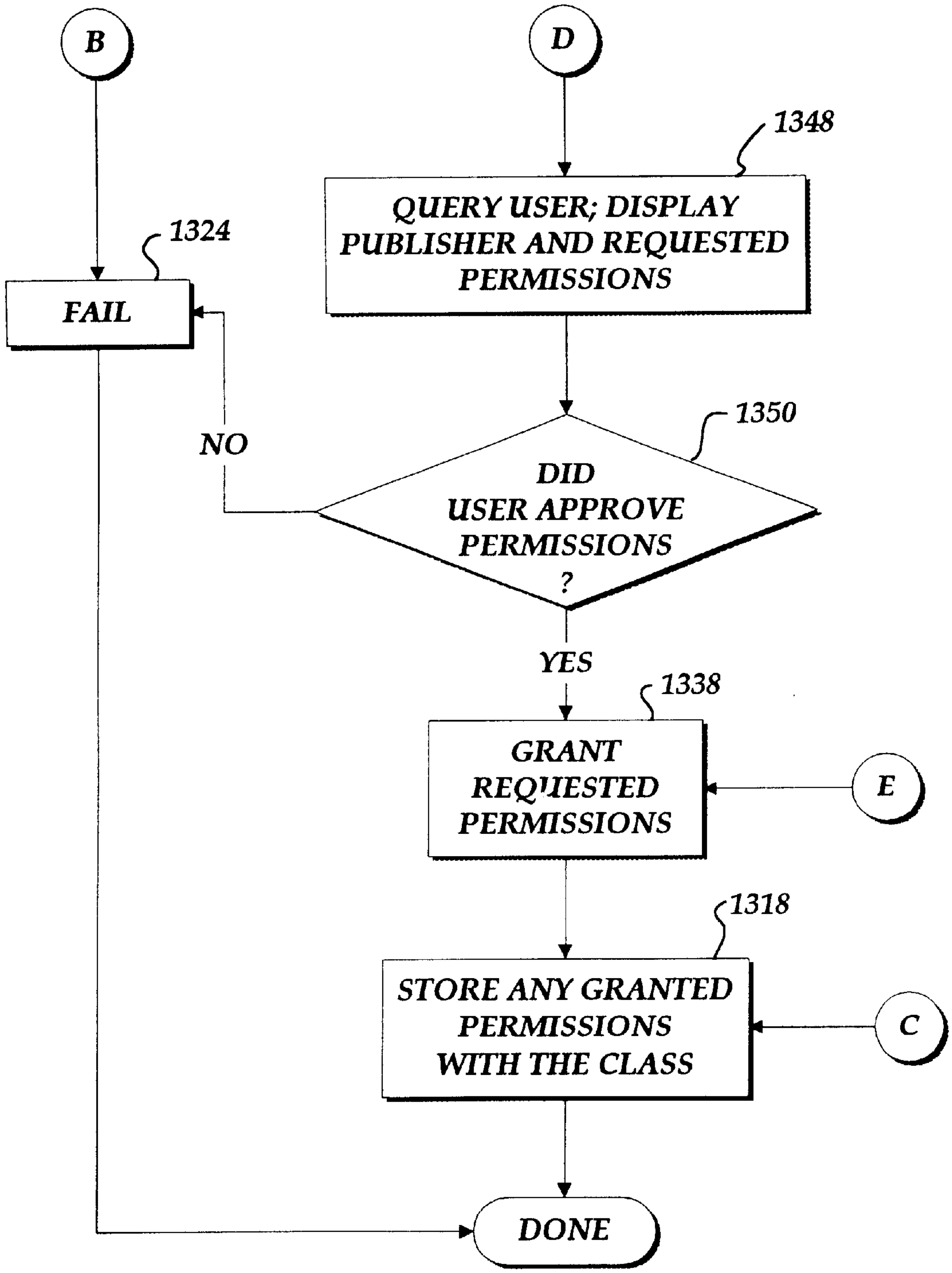

B
D
1348
QUERY USER; DISPLAY PUBLISHER AND REQUESTED PERMISSIONS
1324
FAIL
NO
1350
DID USER APPROVE PERMISSIONS ?
YES
1338
GRANT REQUESTED PERMISSIONS
E
1318
STORE ANY GRANTED PERMISSIONS WITH THE CLASS
C
DONE

Figure 13C

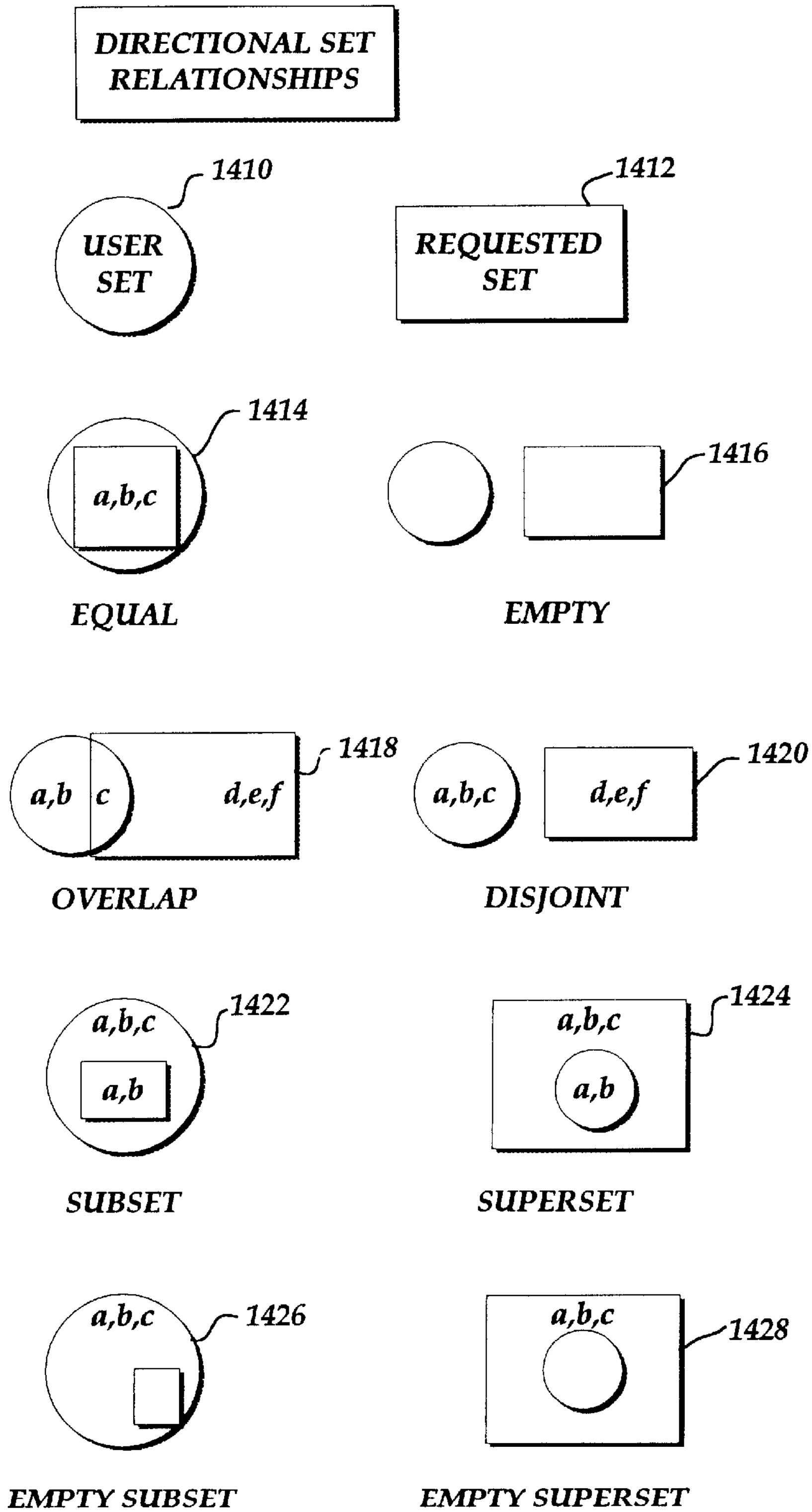

DIRECTIONAL SET RELATIONSHIPS
1410
USER SET
1412
REQUESTED SET
1414
a,b,c
1416
EQUAL
EMPTY
1418
a,b
c
d,e,f
1420
a,b,c
d,e,f
OVERLAP
DISJOINT
1422
a,b,c
a,b
1424
a,b,c
a,b
SUBSET
SUPERSET
1426
a,b,c
1428
a,b,c
EMPTY SUBSET
EMPTY SUPERSET

Figure 14A

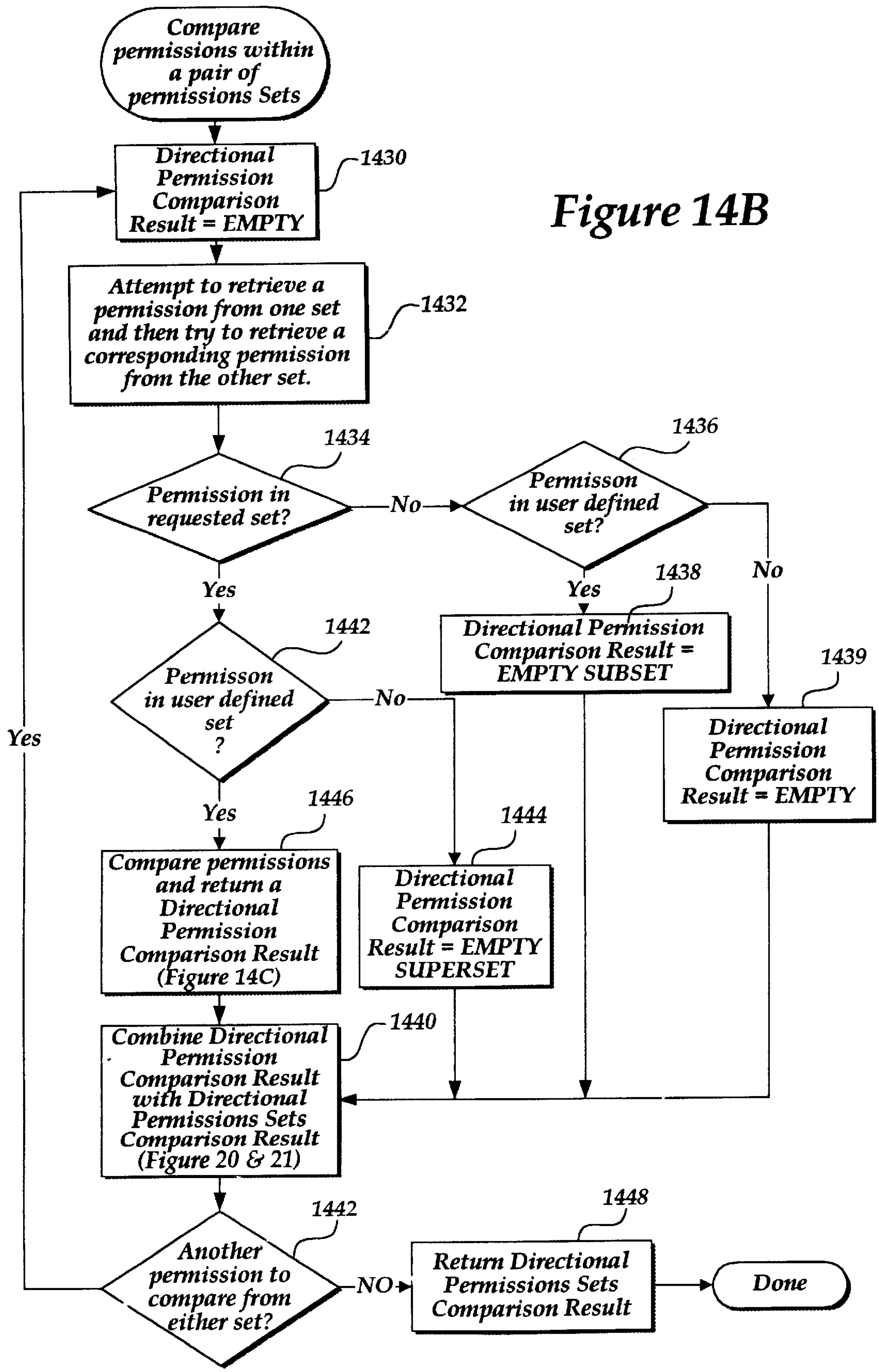

Compare permissions within a pair of permissions Sets
Directional Permission Comparison Result = EMPTY
1430
Attempt to retrieve a permission from one set and then try to retrieve a corresponding permission from the other set.
1432
1434
Permission in requested set?
No
1436
Permisson in user defined set?
No
Yes
Yes
1438
Directional Permission Comparison Result = EMPTY SUBSET
1442
Permisson in user defined set ?
No
1439
Directional Permission Comparison Result = EMPTY
Yes
Yes
1446
1444
Compare permissions and return a Directional Permission Comparison Result (Figure 14C)
Directional Permission Comparison Result = EMPTY SUPERSET
1440
Combine Directional Permission Comparison Result with Directional Permissions Sets Comparison Result (Figure 20 & 21)
1442
1448
Another permission to compare from either set?
NO
Return Directional Permissions Sets Comparison Result
Done

Figure 14B

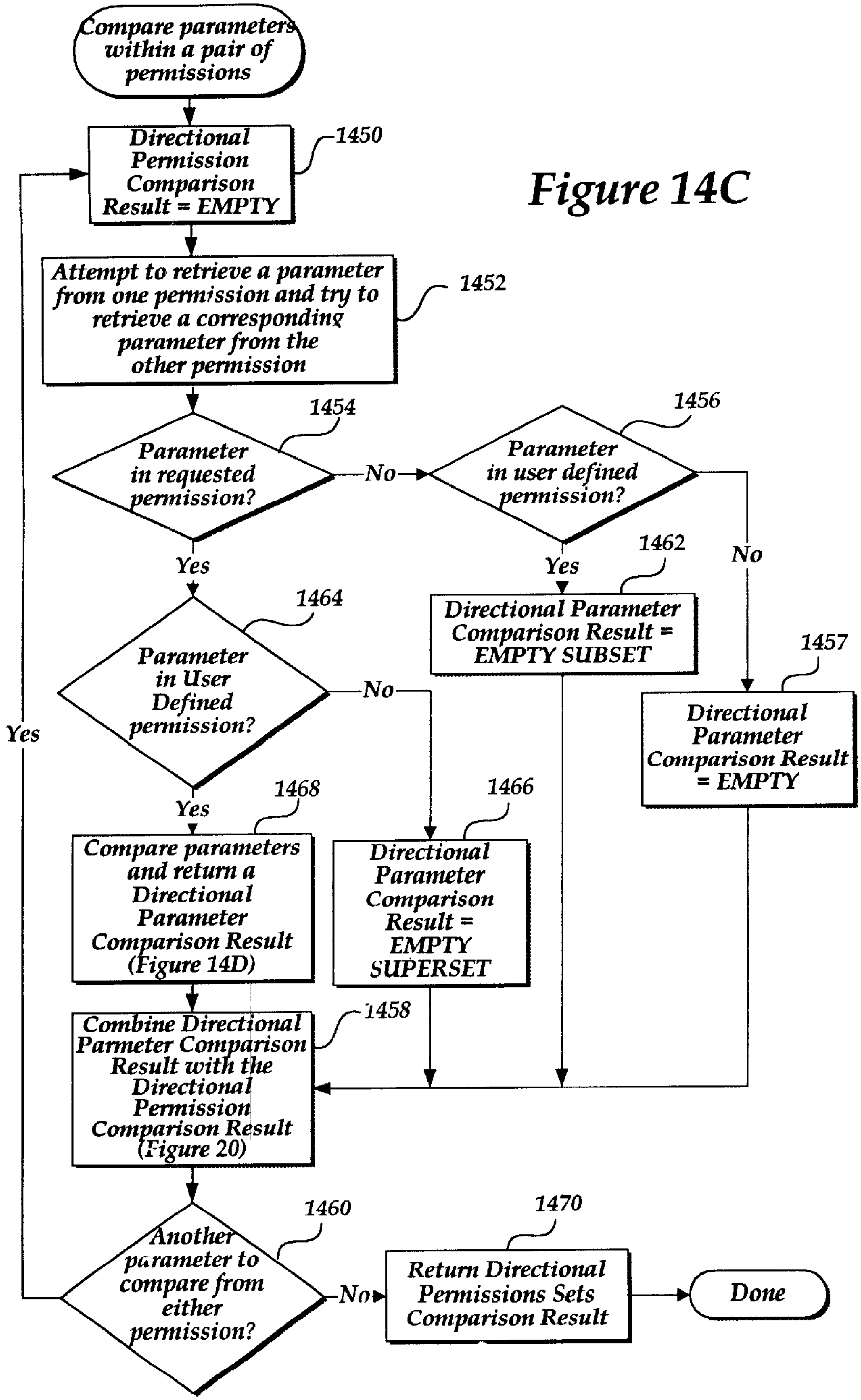
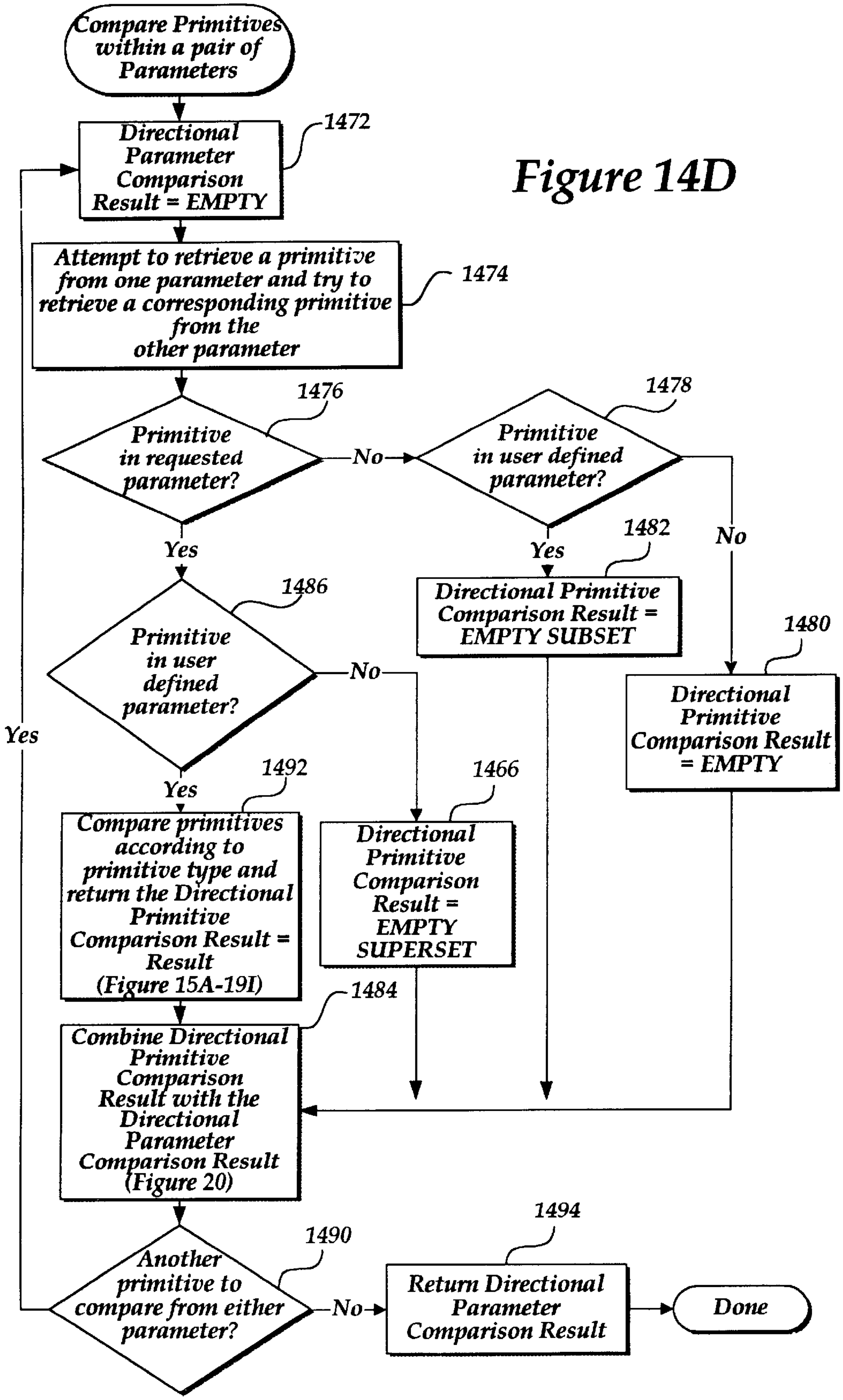

Compare parameters within a pair of permissions
Directional Permission Comparison Result = EMPTY
1450
Attempt to retrieve a parameter from one permission and try to retrieve a corresponding parameter from the other permission
1452
Parameter in requested permission?
1454
No
Parameter in user defined permission?
1456
Yes
No
1462
Directional Parameter Comparison Result = EMPTY SUBSET
Parameter in User Defined permission?
1464
No
1457
Directional Parameter Comparison Result = EMPTY
Yes
Yes
1468
1466
Compare parameters and return a Directional Parameter Comparison Result (Figure 14D)
Directional Parameter Comparison Result = EMPTY SUPERSET
Combine Directional Parmeter Comparison Result with the Directional Permission Comparison Result (Figure 20)
1458
Another parameter to compare from either permission?
1460
No
Return Directional Permissions Sets Comparison Result
1470
Done

Figure 14C

Figure 14D


Compare Primitives within a pair of Parameters
Directional Parameter Comparison Result = EMPTY
1472
Attempt to retrieve a primitive from one parameter and try to retrieve a corresponding primitive from the other parameter
1474
Primitive in requested parameter?
1476
No
Primitive in user defined parameter?
1478
Yes
No
Directional Primitive Comparison Result = EMPTY SUBSET
1482
Yes
Primitive in user defined parameter?
1486
No
Directional Primitive Comparison Result = EMPTY
1480
Yes
Yes
Compare primitives according to primitive type and return the Directional Primitive Comparison Result = Result (Figure 15A-19I)
1492
Directional Primitive Comparison Result = EMPTY SUPERSET
1466
Combine Directional Primitive Comparison Result with the Directional Parameter Comparison Result (Figure 20)
1484
Another primitive to compare from either parameter?
1490
No
Return Directional Parameter Comparison Result
1494
Done

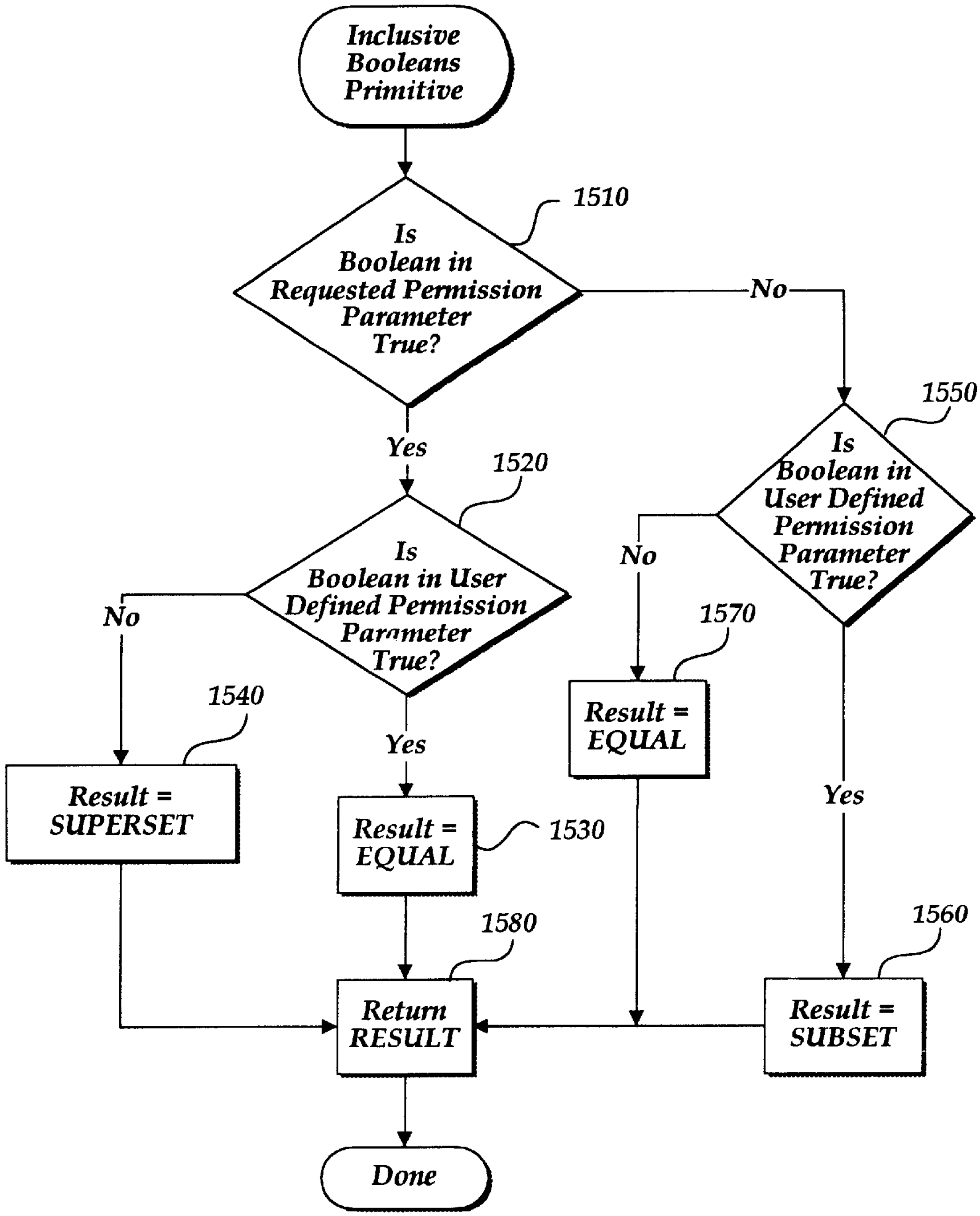

Inclusive Booleans Primitive
1510
Is Boolean in Requested Permission Parameter True?
No
Yes
1520
Is Boolean in User Defined Permission Parameter True?
No
Yes
1540
Result = SUPERSET
Result = EQUAL
1530
1550
Is Boolean in User Defined Permission Parameter True?
No
Yes
1570
Result = EQUAL
1560
Result = SUBSET
1580
Return RESULT
Done

Figure 15A

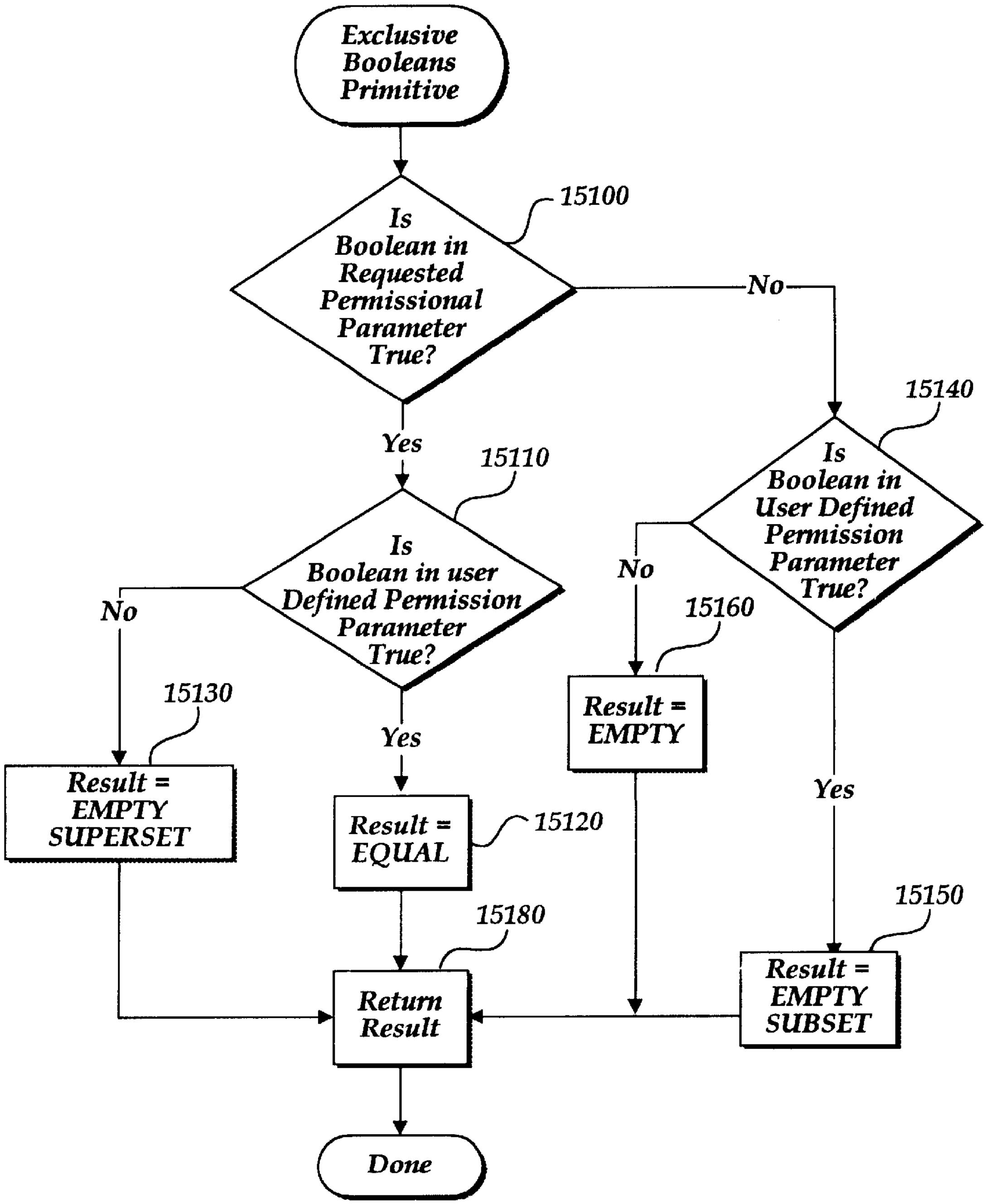

Exclusive Booleans Primitive
Is Boolean in Requested Permissional Parameter True?
15100
No
Yes
15140
Is Boolean in User Defined Permission Parameter True?
15110
Is Boolean in user Defined Permission Parameter True?
No
No
15160
15130
Yes
Result = EMPTY
Result = EMPTY SUPERSET
Result = EQUAL
15120
Yes
15150
15180
Return Result
Result = EMPTY SUBSET
Done

Figure 15B

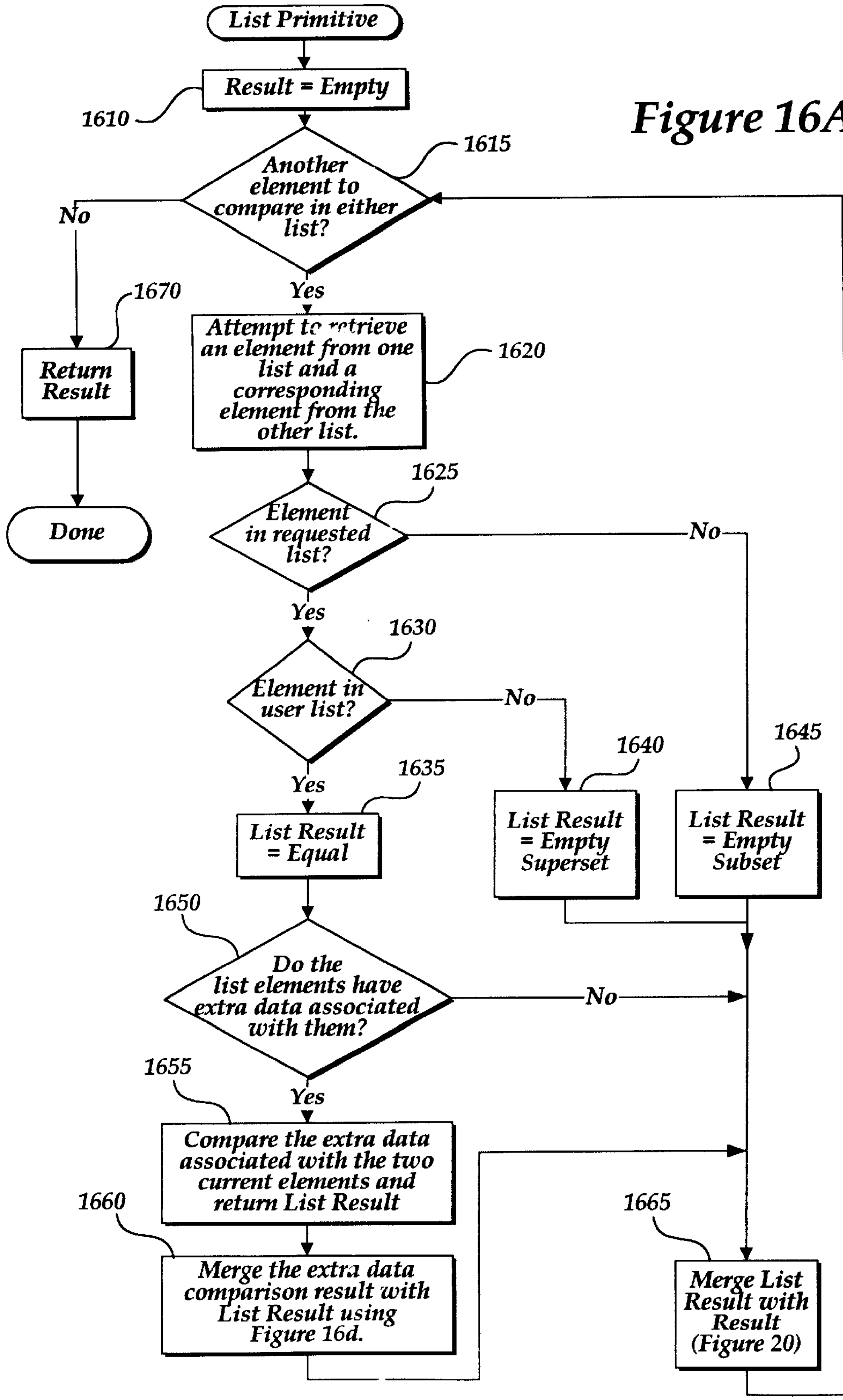

List Primitive
Result = Empty
1610
Another element to compare in either list?
1615
No
Yes
1670
Return Result
Done
Attempt to retrieve an element from one list and a corresponding element from the other list.
1620
1625
Element in requested list?
No
Yes
1630
Element in user list?
No
Yes
1635
List Result = Equal
1640
List Result = Empty Superset
1645
List Result = Empty Subset
1650
Do the list elements have extra data associated with them?
No
1655
Yes
Compare the extra data associated with the two current elements and return List Result
1660
Merge the extra data comparison result with List Result using Figure 16d.
1665
Merge List Result with Result (Figure 20)

Figure 16A

	EMPTY	EMPTY SUBSET	EMPTY SUPERSET	OVERLAP	DISJOINT	SUBSET	EQUAL	SUPERSET
OVERLAP	OVERLAP	OVERLAP	OVERLAP	OVERLAP	DISJOINT	OVERLAP	OVERLAP	OVERLAP
DISJOINT	DISJOINT	DISJOINT	DISJOINT	DISJOINT	DISJOINT	DISJOINT	DISJOINT	DISJOINT
SUBSET	SUBSET	SUBSET	DISJOINT	OVERLAP	DISJOINT	SUBSET	SUBSET	DISJOINT
EQUAL	EQUAL	SUBSET	SUPERSET	OVERLAP	DISJOINT	SUBSET	EQUAL	SUPERSET
SUPERSET	SUPERSET	DISJOINT	SUPERSET	OVERLAP	DISJOINT	DISJOINT	SUPERSET	SUPERSET

FIGURE 16B

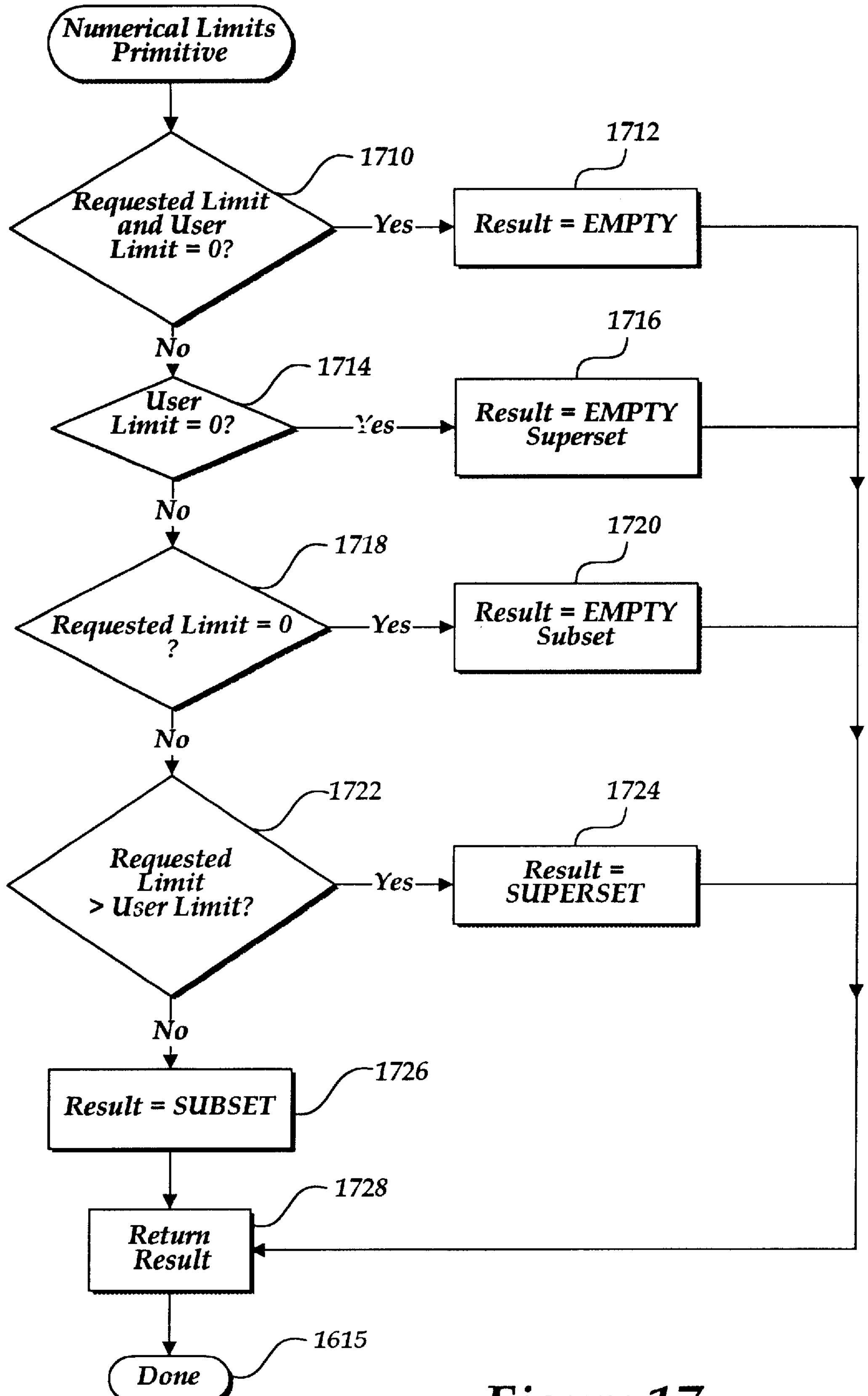

Numerical Limits Primitive
Requested Limit and User Limit = 0?
1710
Yes
No
Result = EMPTY
1712
User Limit = 0?
1714
Yes
No
Result = EMPTY Superset
1716
Requested Limit = 0 ?
1718
Yes
No
Result = EMPTY Subset
1720
Requested Limit > User Limit?
1722
Yes
No
Result = SUPERSET
1724
Result = SUBSET
1726
Return Result
1728
Done
1615

Figure 17

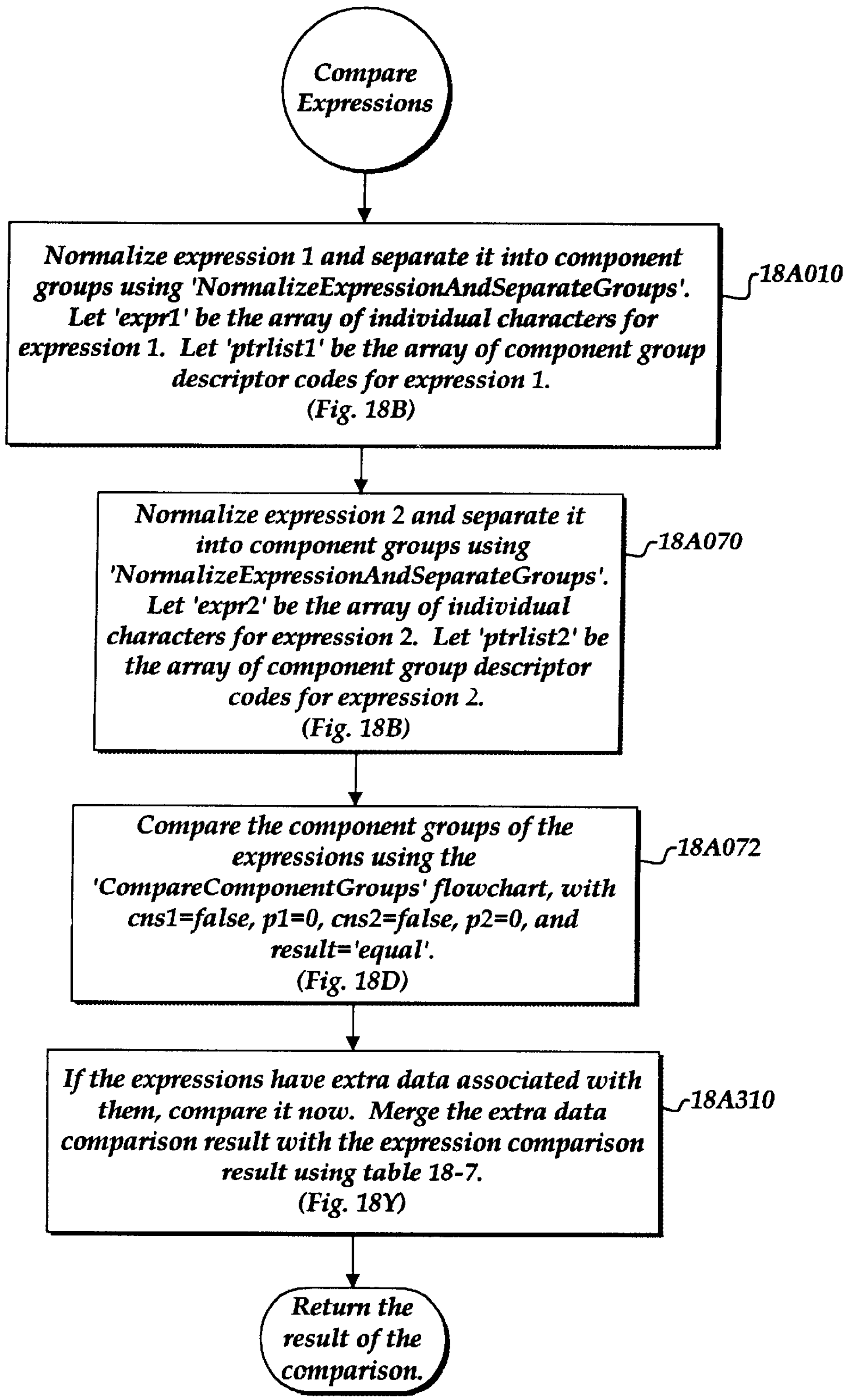

Compare Expressions
Normalize expression 1 and separate it into component groups using 'NormalizeExpressionAndSeparateGroups'. Let 'expr1' be the array of individual characters for expression 1. Let 'ptrlist1' be the array of component group descriptor codes for expression 1. (Fig. 18B)
18A010
Normalize expression 2 and separate it into component groups using 'NormalizeExpressionAndSeparateGroups'. Let 'expr2' be the array of individual characters for expression 2. Let 'ptrlist2' be the array of component group descriptor codes for expression 2. (Fig. 18B)
18A070
Compare the component groups of the expressions using the 'CompareComponentGroups' flowchart, with cns1=false, p1=0, cns2=false, p2=0, and result='equal'. (Fig. 18D)
18A072
If the expressions have extra data associated with them, compare it now. Merge the extra data comparison result with the expression comparison result using table 18-7. (Fig. 18Y)
18A310
Return the result of the comparison.

Figure 18A

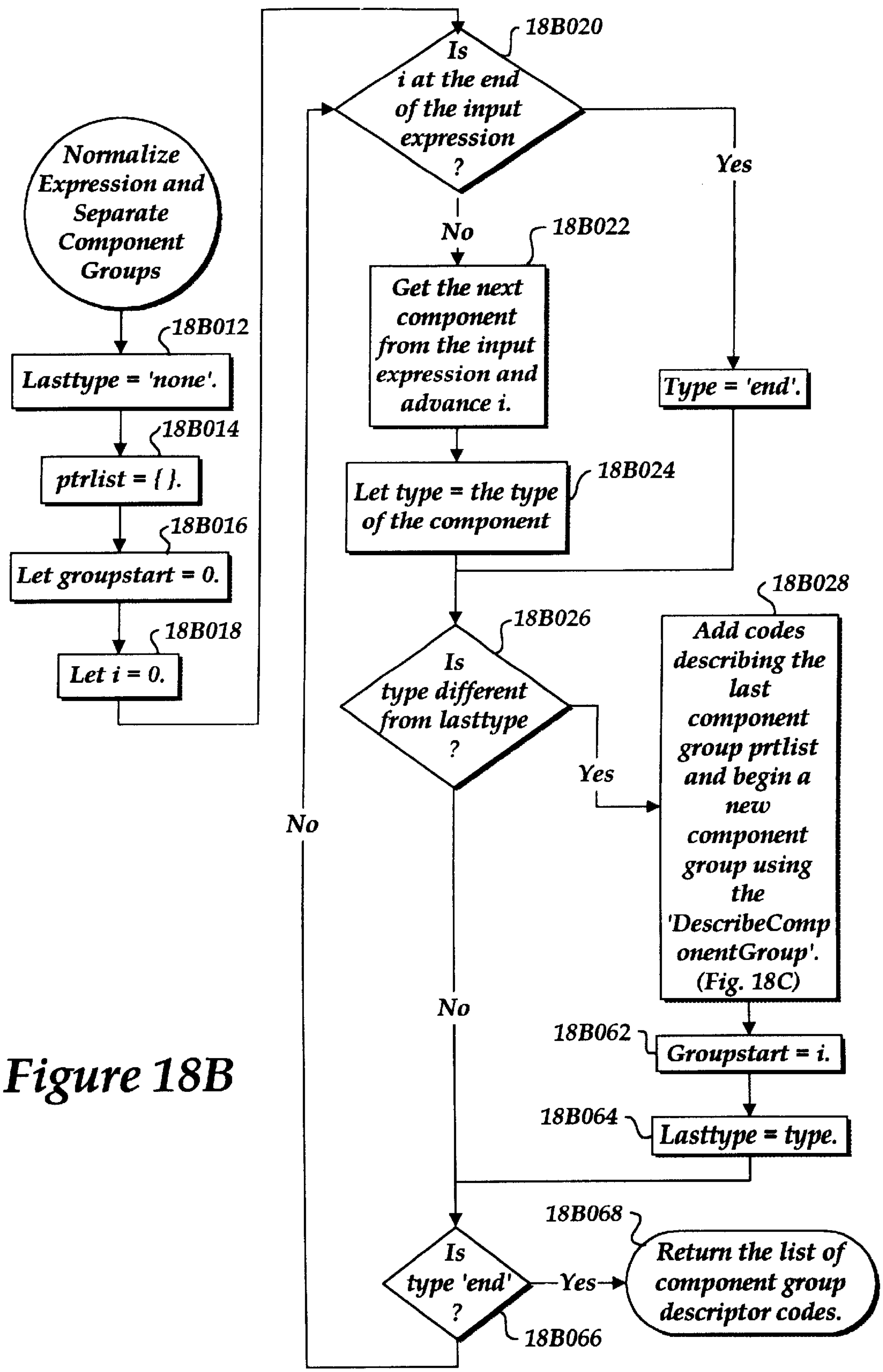

Normalize Expression and Separate Component Groups
18B012
Lasttype = 'none'.
18B014
ptrlist = { }.
18B016
Let groupstart = 0.
18B018
Let i = 0.
18B020
Is i at the end of the input expression ?
Yes
No
18B022
Get the next component from the input expression and advance i.
Type = 'end'.
18B024
Let type = the type of the component
18B028
Add codes describing the last component group prtlist and begin a new component group using the 'DescribeComponentGroup'. (Fig. 18C)
18B026
Is type different from lasttype ?
Yes
No
No
18B062
Groupstart = i.
18B064
Lasttype = type.
18B068
Is type 'end' ?
Yes
Return the list of component group descriptor codes.
18B066

Figure 18B

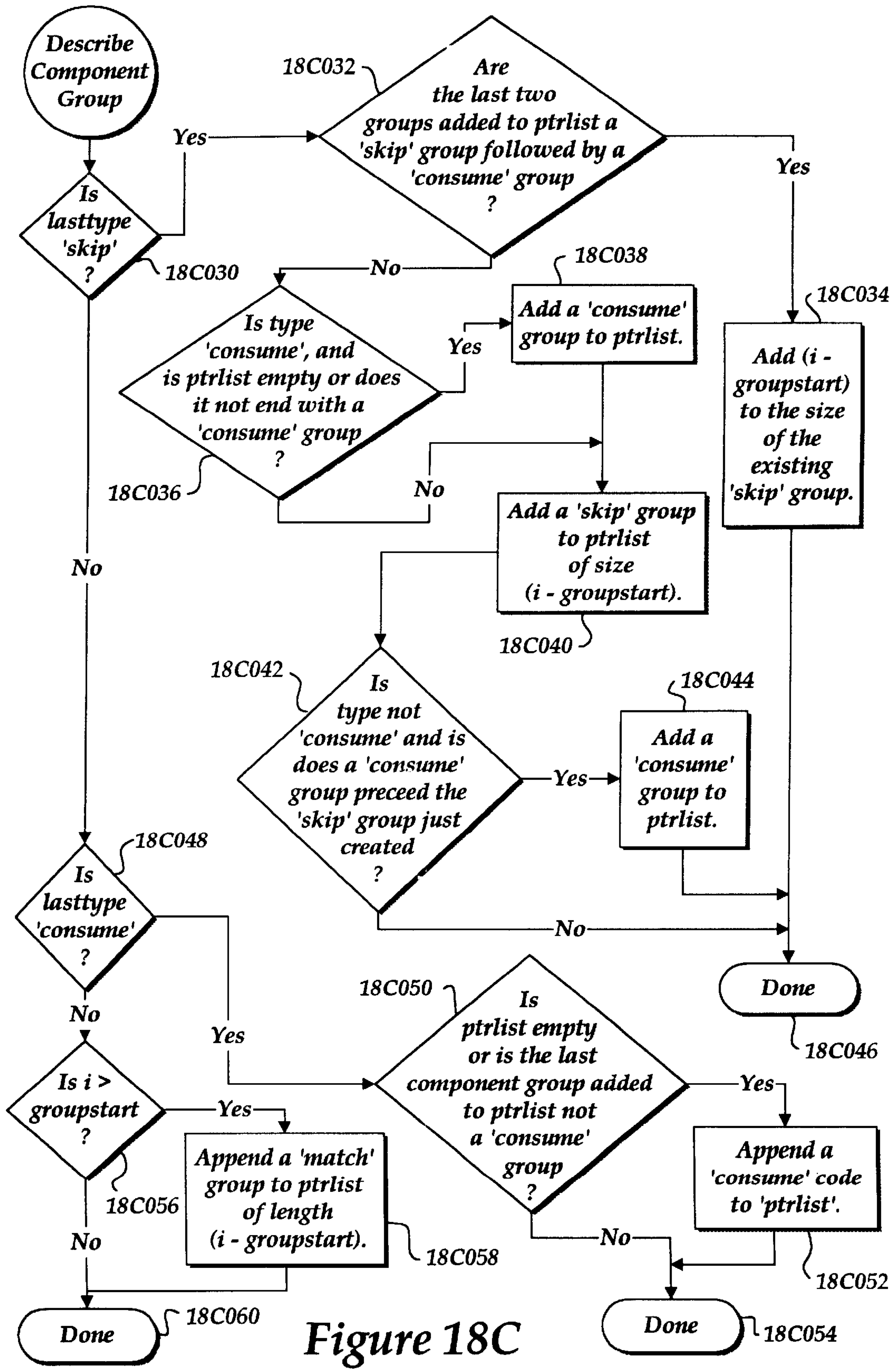
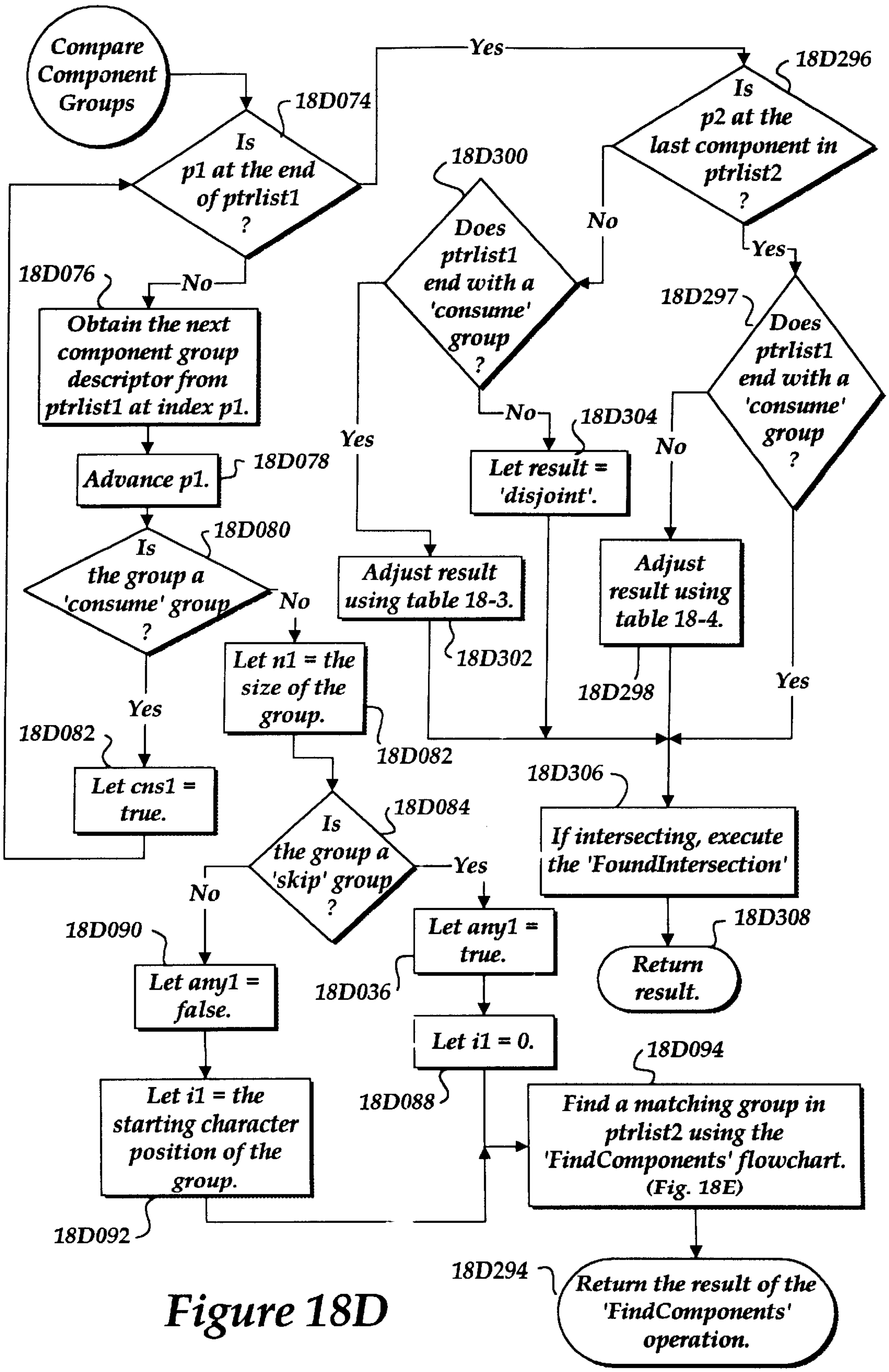

Describe Component Group
Is lasttype 'skip' ?
18C030
Yes
No
Are the last two groups added to ptrlist a 'skip' group followed by a 'consume' group ?
18C032
Yes
No
Is type 'consume', and is ptrlist empty or does it not end with a 'consume' group ?
18C036
Yes
No
Add a 'consume' group to ptrlist.
18C038
Add a 'skip' group to ptrlist of size (i - groupstart).
18C040
Add (i - groupstart) to the size of the existing 'skip' group.
18C034
Is type not 'consume' and is does a 'consume' group preceed the 'skip' group just created ?
18C042
Yes
No
Add a 'consume' group to ptrlist.
18C044
Done
18C046
Is lasttype 'consume' ?
18C048
Yes
No
Is ptrlist empty or is the last component group added to ptrlist not a 'consume' group ?
18C050
Yes
No
Append a 'consume' code to 'ptrlist'.
18C052
Done
18C054
Is i > groupstart ?
18C056
Yes
No
Append a 'match' group to ptrlist of length (i - groupstart).
18C058
Done
18C060

Figure 18C



Compare Component Groups
Is p1 at the end of ptrlist1 ?
18D074
Yes
No
18D076
Obtain the next component group descriptor from ptrlist1 at index p1.
Advance p1.
18D078
Is the group a 'consume' group ?
18D080
No
Yes
18D082
Let cns1 = true.
Let n1 = the size of the group.
18D082
Is the group a 'skip' group ?
18D084
No
Yes
18D090
Let any1 = false.
Let i1 = the starting character position of the group.
18D092
Let any1 = true.
18D036
Let i1 = 0.
18D088
Find a matching group in ptrlist2 using the 'FindComponents' flowchart. (Fig. 18E)
18D094
Return the result of the 'FindComponents' operation.
18D294
Is p2 at the last component in ptrlist2 ?
18D296
No
Yes
Does ptrlist1 end with a 'consume' group ?
18D300
Yes
No
Let result = 'disjoint'.
18D304
Adjust result using table 18-3.
18D302
Does ptrlist1 end with a 'consume' group ?
18D297
No
Yes
Adjust result using table 18-4.
18D298
If intersecting, execute the 'FoundIntersection'
18D306
Return result.
18D308

Figure 18D

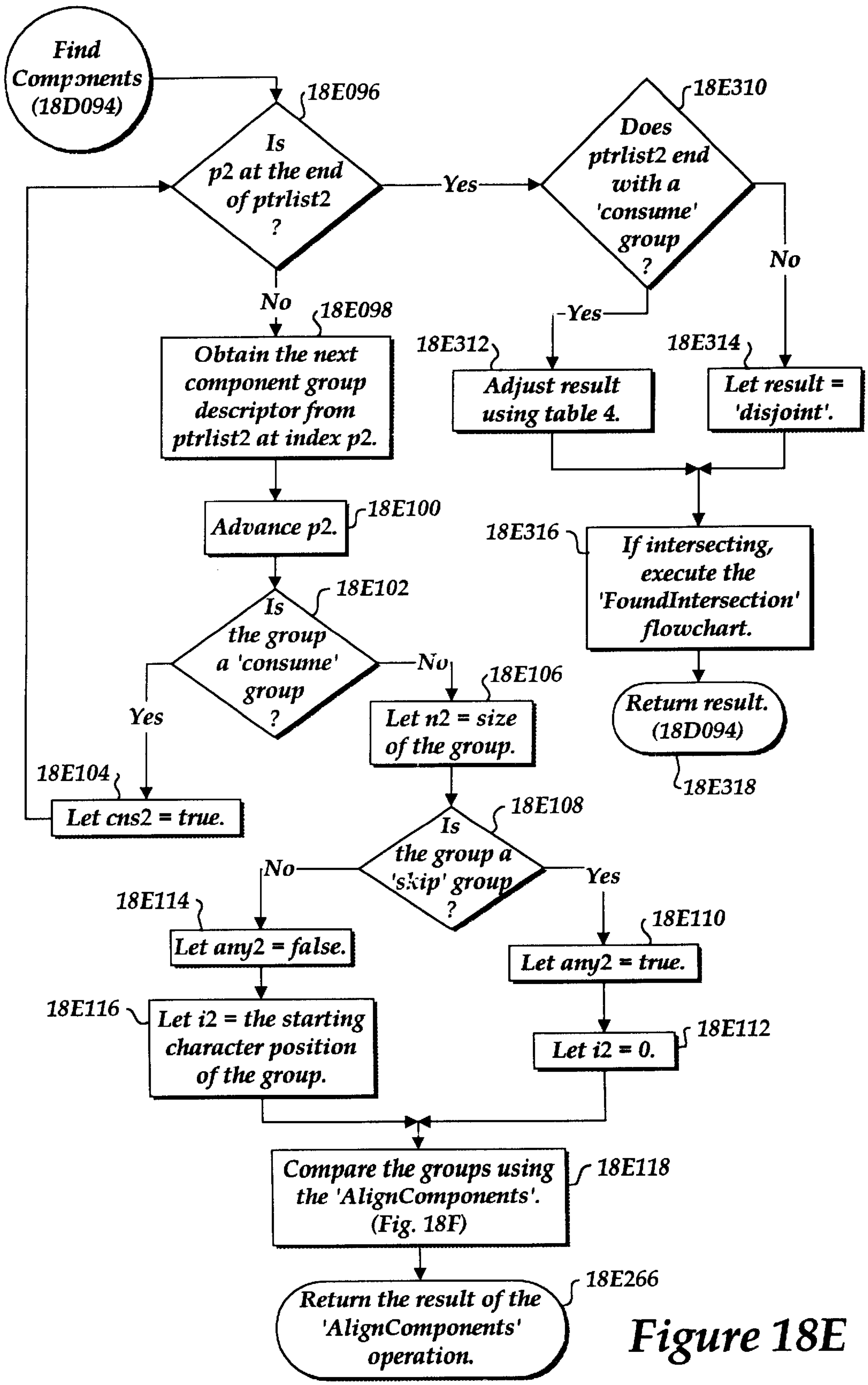

Find Components (18D094)
18E096
Is p2 at the end of ptrlist2 ?
Yes
No
18E098
Obtain the next component group descriptor from ptrlist2 at index p2.
18E100
Advance p2.
18E102
Is the group a 'consume' group ?
Yes
No
18E104
Let cns2 = true.
18E106
Let n2 = size of the group.
18E108
Is the group a 'skip' group ?
No
Yes
18E114
Let any2 = false.
18E116
Let i2 = the starting character position of the group.
18E110
Let any2 = true.
18E112
Let i2 = 0.
Compare the groups using the 'AlignComponents'. (Fig. 18F)
18E118
Return the result of the 'AlignComponents' operation.
18E266
18E310
Does ptrlist2 end with a 'consume' group ?
Yes
No
18E312
Adjust result using table 4.
18E314
Let result = 'disjoint'.
18E316
If intersecting, execute the 'FoundIntersection' flowchart.
Return result. (18D094)
18E318

Figure 18E

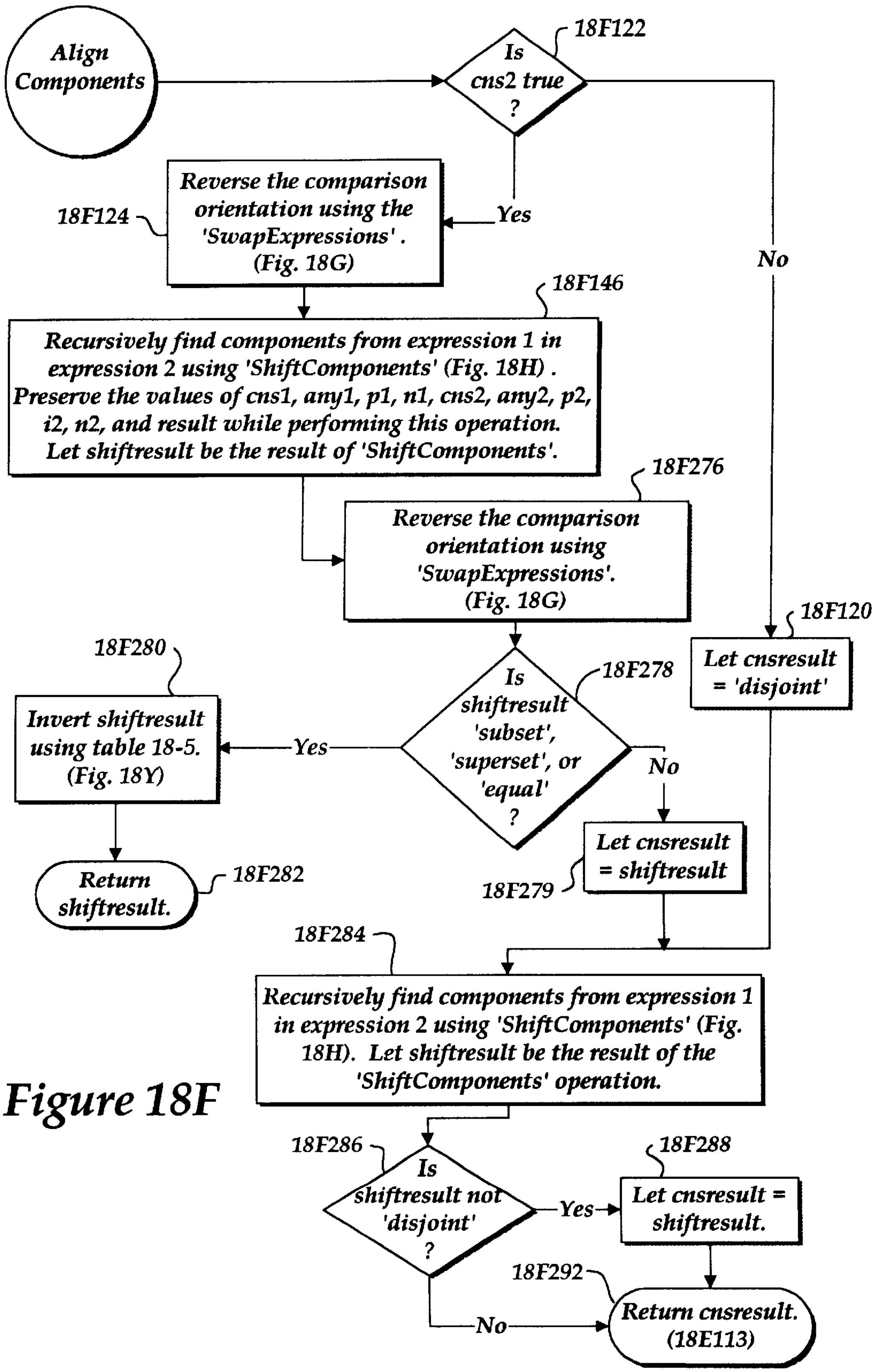

Align Components
18F122
Is cns2 true ?
Yes
No
18F124
Reverse the comparison orientation using the 'SwapExpressions'. (Fig. 18G)
18F146
Recursively find components from expression 1 in expression 2 using 'ShiftComponents' (Fig. 18H). Preserve the values of cns1, any1, p1, n1, cns2, any2, p2, i2, n2, and result while performing this operation. Let shiftresult be the result of 'ShiftComponents'.
18F276
Reverse the comparison orientation using 'SwapExpressions'. (Fig. 18G)
18F120
Let cnsresult = 'disjoint'
18F280
Invert shiftresult using table 18-5. (Fig. 18Y)
Yes
18F278
Is shiftresult 'subset', 'superset', or 'equal' ?
No
Let cnsresult = shiftresult
18F279
Return shiftresult.
18F282
18F284
Recursively find components from expression 1 in expression 2 using 'ShiftComponents' (Fig. 18H). Let shiftresult be the result of the 'ShiftComponents' operation.
18F286
Is shiftresult not 'disjoint' ?
Yes
18F288
Let cnsresult = shiftresult.
18F292
No
Return cnsresult. (18E113)

Figure 18F

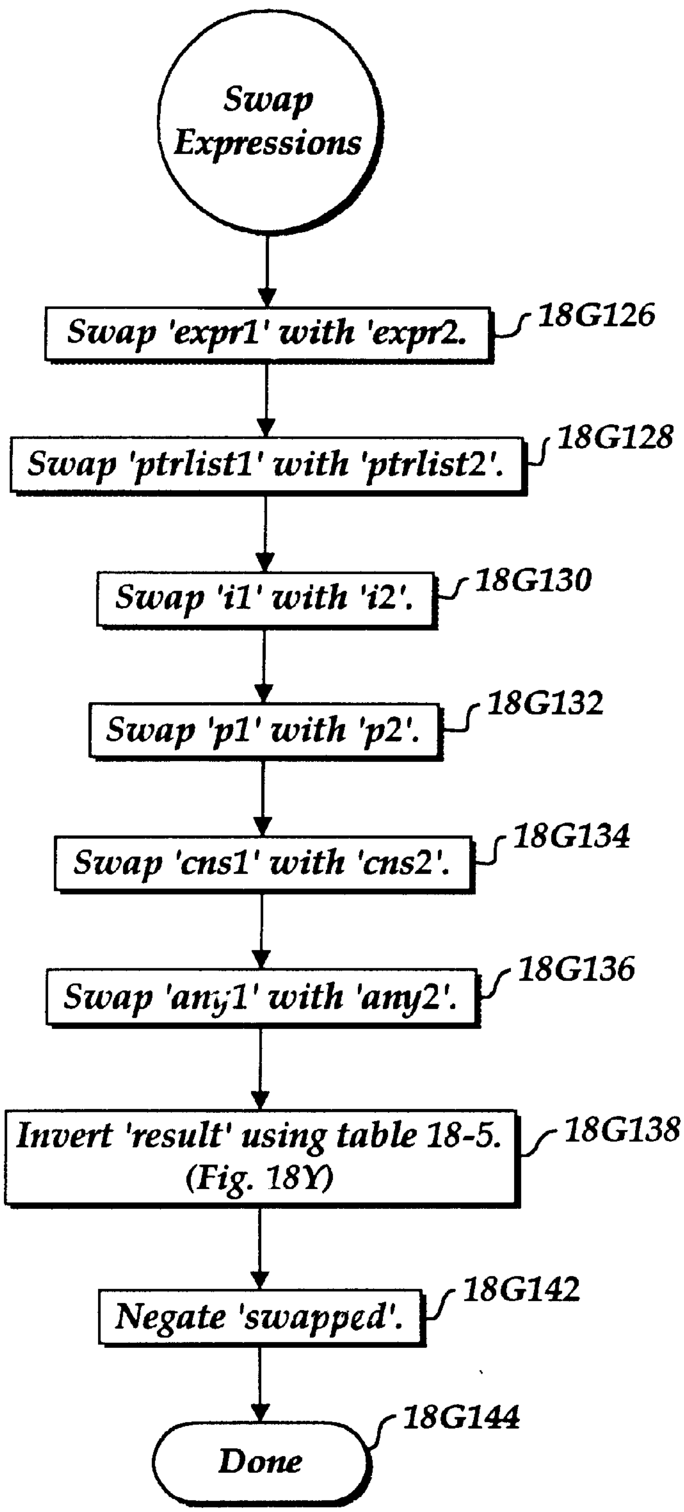

Swap Expressions
Swap 'expr1' with 'expr2.
18G126
Swap 'ptrlist1' with 'ptrlist2'.
18G128
Swap 'i1' with 'i2'.
18G130
Swap 'p1' with 'p2'.
18G132
Swap 'cns1' with 'cns2'.
18G134
Swap 'any1' with 'any2'.
18G136
Invert 'result' using table 18-5. (Fig. 18Y)
18G138
Negate 'swapped'.
18G142
Done
18G144

Figure 18G

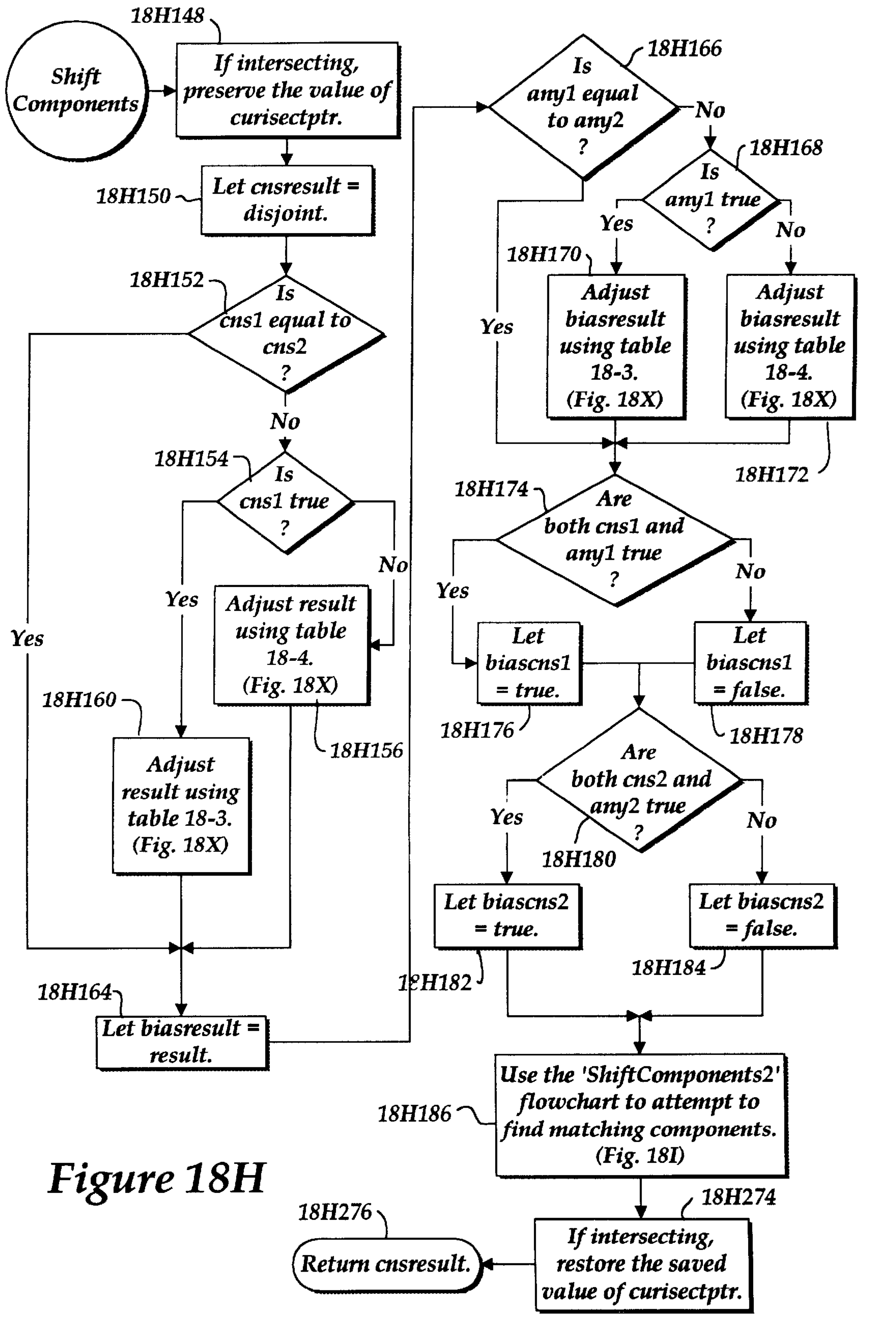
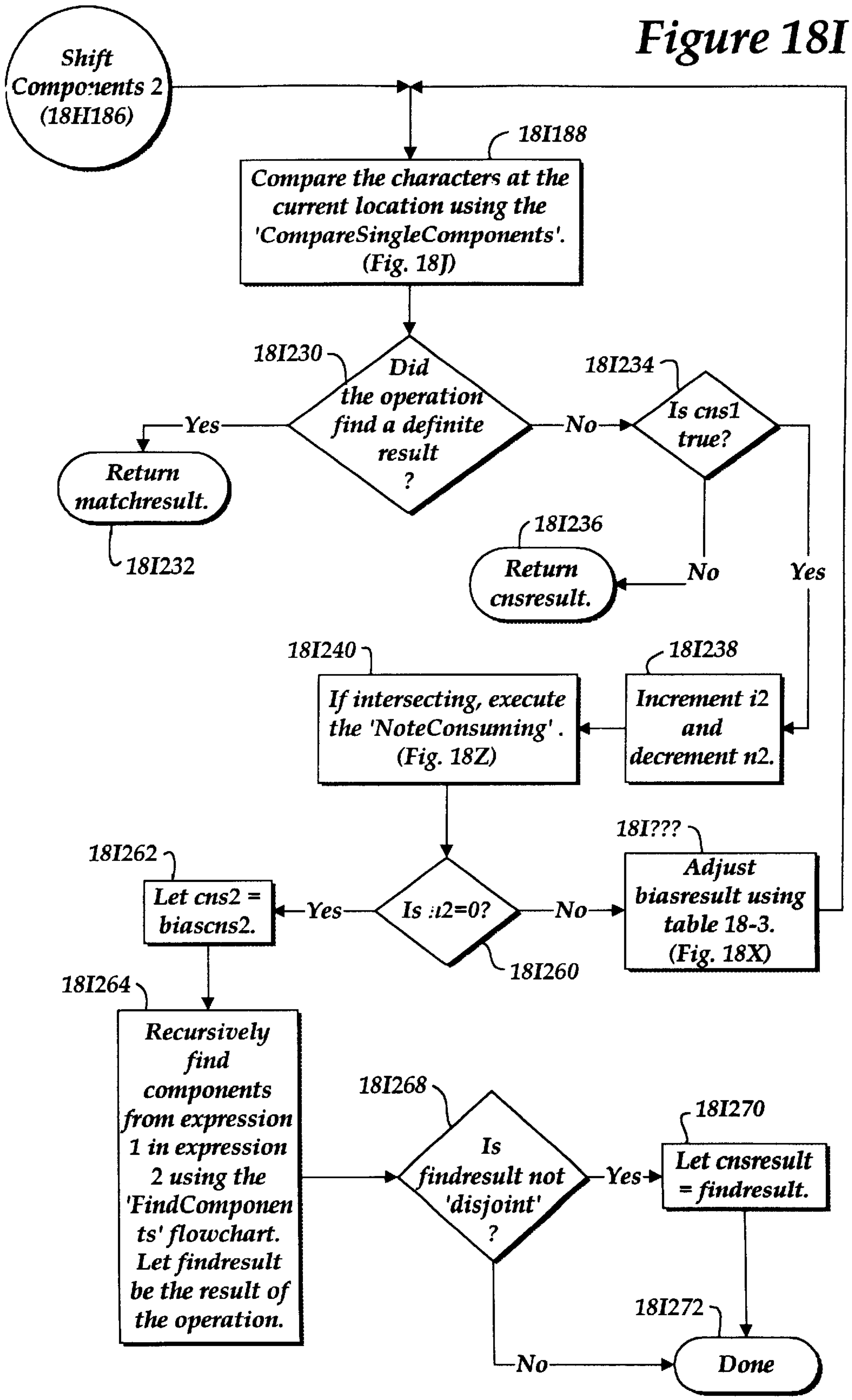

Shift Components
18H148
If intersecting, preserve the value of curisectptr.
18H150
Let cnsresult = disjoint.
18H152
Is cns1 equal to cns2 ?
Yes
No
18H154
Is cns1 true ?
Yes
No
Adjust result using table 18-4. (Fig. 18X)
18H156
18H160
Adjust result using table 18-3. (Fig. 18X)
18H164
Let biasresult = result.
18H166
Is any1 equal to any2 ?
No
Yes
18H168
Is any1 true ?
Yes
No
18H170
Adjust biasresult using table 18-3. (Fig. 18X)
Adjust biasresult using table 18-4. (Fig. 18X)
18H172
18H174
Are both cns1 and any1 true ?
Yes
No
Let biascns1 = true.
Let biascns1 = false.
18H176
18H178
Are both cns2 and any2 true ?
Yes
No
18H180
Let biascns2 = true.
Let biascns2 = false.
18H182
18H184
Use the 'ShiftComponents2' flowchart to attempt to find matching components. (Fig. 18I)
18H186
18H274
If intersecting, restore the saved value of curisectptr.
18H276
Return cnsresult.

Figure 18H

Figure 18I


Shift Components 2 (18H186)
18I188
Compare the characters at the current location using the 'CompareSingleComponents'. (Fig. 18J)
18I230
Did the operation find a definite result ?
Yes
Return matchresult.
18I232
No
18I234
Is cns1 true?
18I236
Return cnsresult.
No
Yes
18I240
If intersecting, execute the 'NoteConsuming' . (Fig. 18Z)
18I238
Increment i2 and decrement n2.
18I???
Adjust biasresult using table 18-3. (Fig. 18X)
18I262
Let cns2 = biascns2.
Yes
Is n2=0?
No
18I260
18I264
Recursively find components from expression 1 in expression 2 using the 'FindComponents' flowchart. Let findresult be the result of the operation.
18I268
Is findresult not 'disjoint' ?
Yes
18I270
Let cnsresult = findresult.
18I272
No
Done

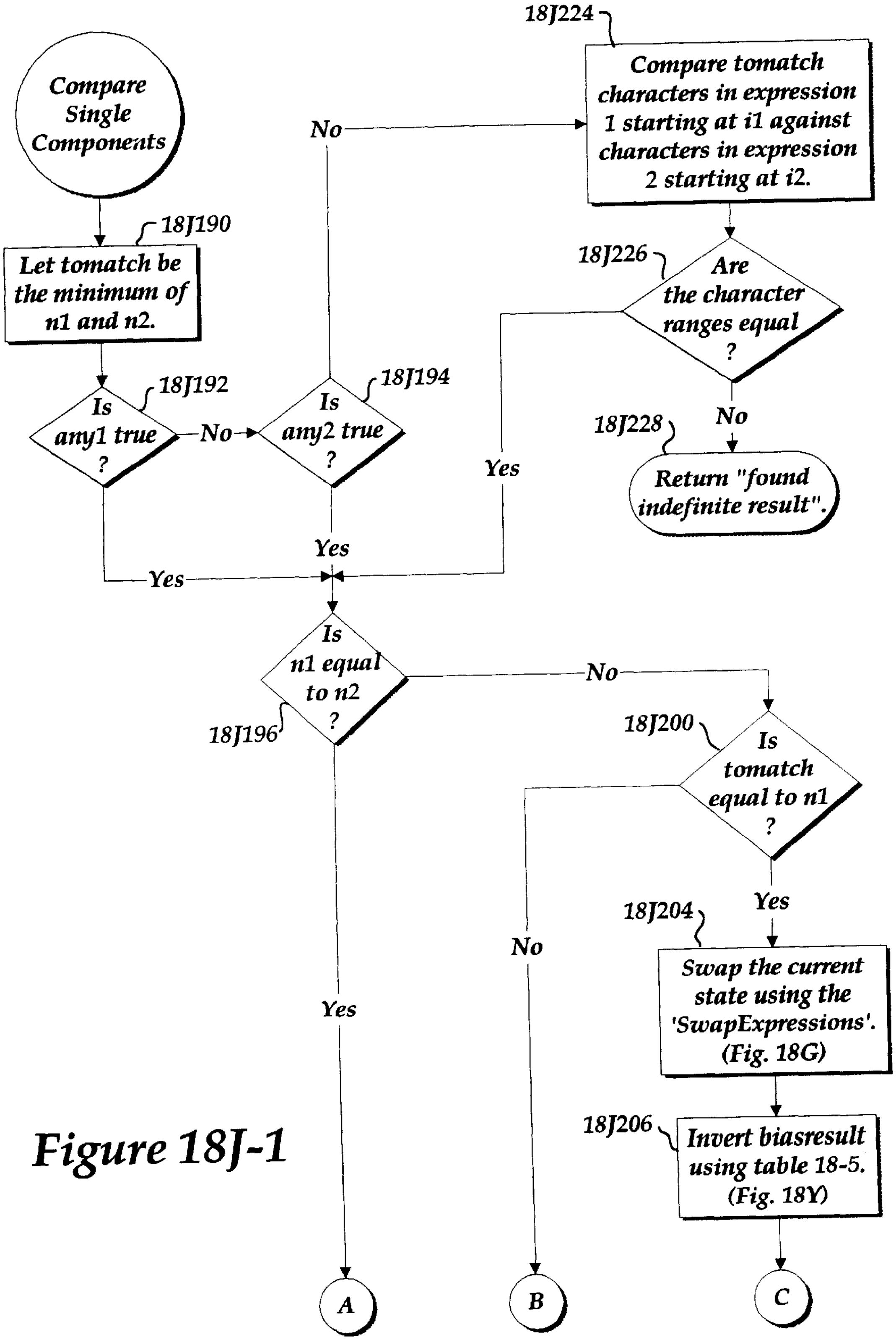
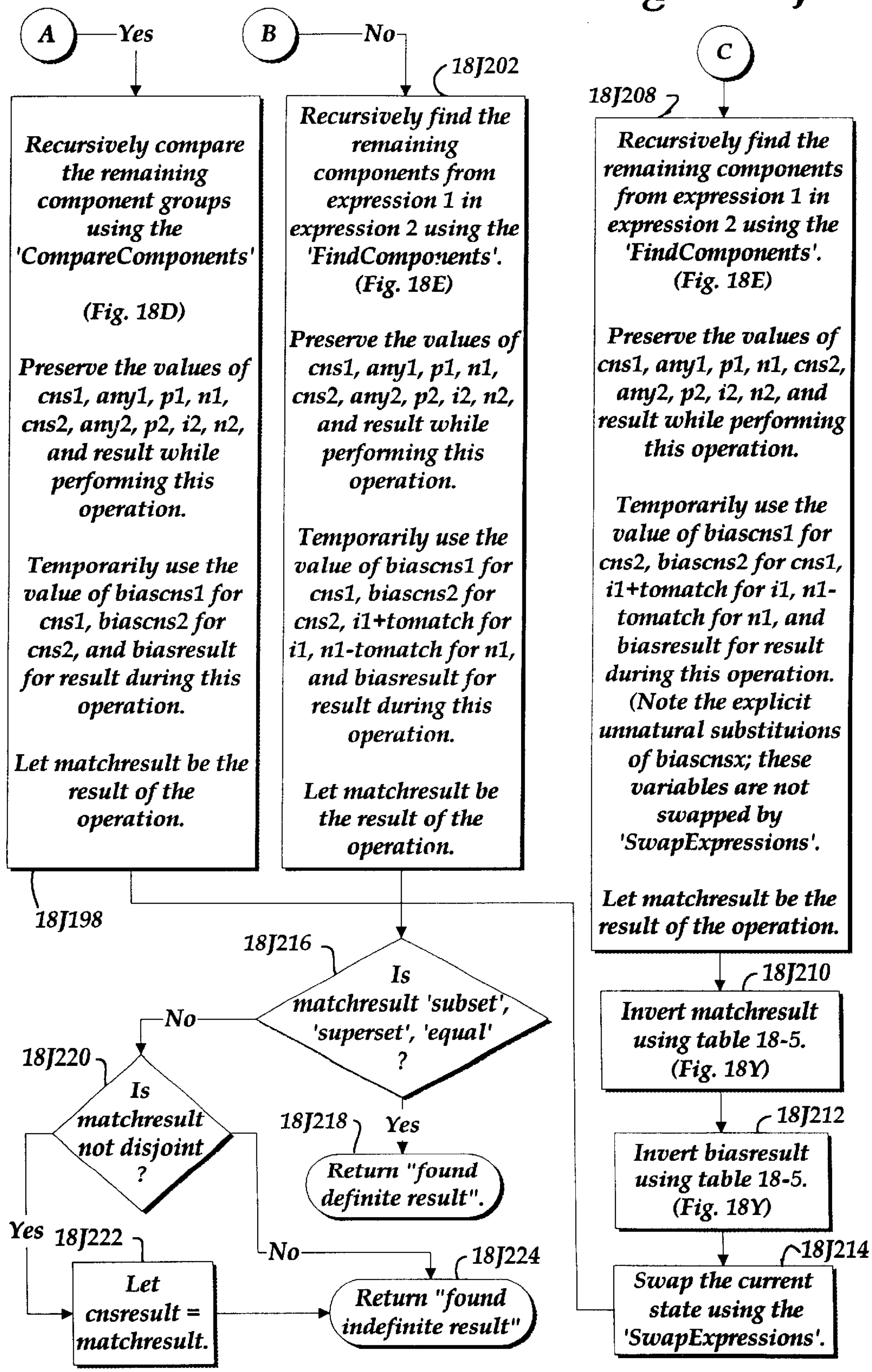

Compare Single Components
18J190
Let tomatch be the minimum of n1 and n2.
18J192
Is any1 true ?
No
18J194
Is any2 true ?
No
Yes
Yes
18J224
Compare tomatch characters in expression 1 starting at i1 against characters in expression 2 starting at i2.
18J226
Are the character ranges equal ?
Yes
No
18J228
Return "found indefinite result".
Is n1 equal to n2 ?
18J196
No
Yes
18J200
Is tomatch equal to n1 ?
No
Yes
18J204
Swap the current state using the 'SwapExpressions'. (Fig. 18G)
18J206
Invert biasresult using table 18-5. (Fig. 18Y)
A
B
C

Figure 18J-1

Figure 18J-2


A
Yes
B
No
C
18J202
18J208
Recursively compare the remaining component groups using the 'CompareComponents'
(Fig. 18D)
Preserve the values of cns1, any1, p1, n1, cns2, any2, p2, i2, n2, and result while performing this operation.
Temporarily use the value of biascns1 for cns1, biascns2 for cns2, and biasresult for result during this operation.
Let matchresult be the result of the operation.
Recursively find the remaining components from expression 1 in expression 2 using the 'FindComponents'. (Fig. 18E)
Preserve the values of cns1, any1, p1, n1, cns2, any2, p2, i2, n2, and result while performing this operation.
Temporarily use the value of biascns1 for cns1, biascns2 for cns2, i1+tomatch for i1, n1-tomatch for n1, and biasresult for result during this operation.
Let matchresult be the result of the operation.
Recursively find the remaining components from expression 1 in expression 2 using the 'FindComponents'. (Fig. 18E)
Preserve the values of cns1, any1, p1, n1, cns2, any2, p2, i2, n2, and result while performing this operation.
Temporarily use the value of biascns1 for cns2, biascns2 for cns1, i1+tomatch for i1, n1-tomatch for n1, and biasresult for result during this operation. (Note the explicit unnatural substituions of biascnsx; these variables are not swapped by 'SwapExpressions'.
Let matchresult be the result of the operation.
18J198
18J216
Is matchresult 'subset', 'superset', 'equal' ?
No
18J220
Is matchresult not disjoint ?
18J218
Yes
Return "found definite result".
18J210
Invert matchresult using table 18-5. (Fig. 18Y)
18J212
Invert biasresult using table 18-5. (Fig. 18Y)
Yes
18J222
No
18J224
18J214
Let cnsresult = matchresult.
Return "found indefinite result"
Swap the current state using the 'SwapExpressions'.

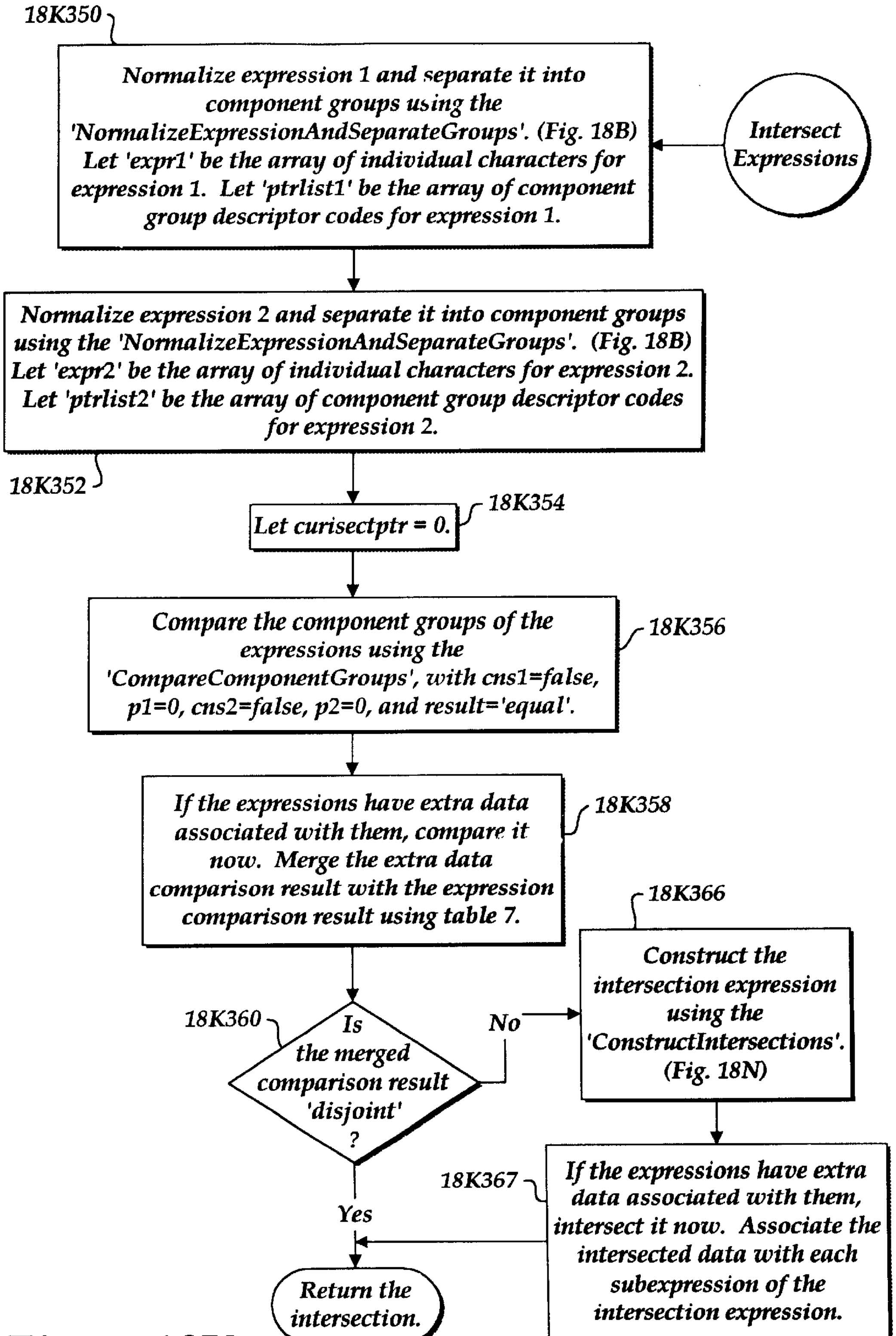
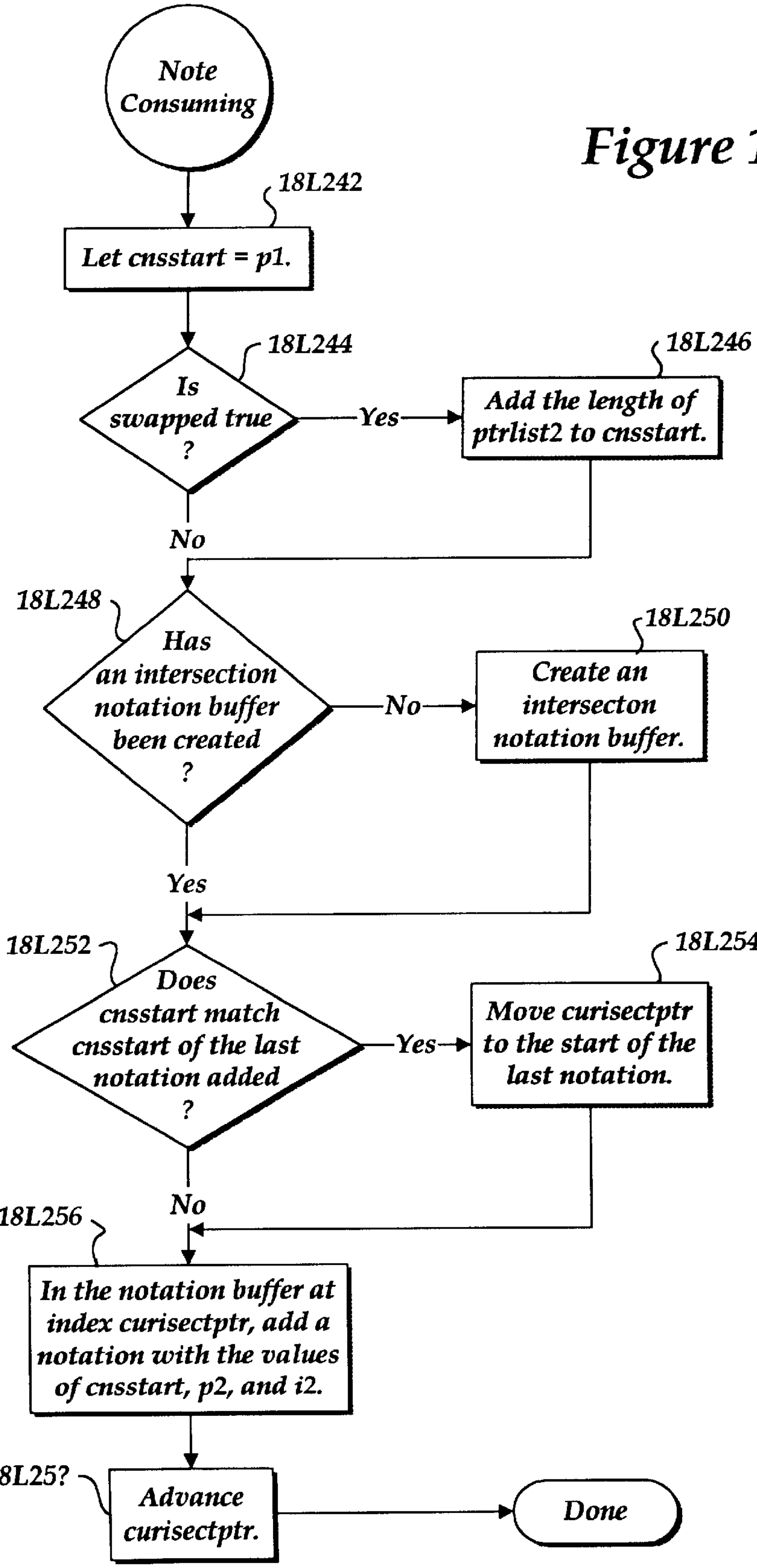

18K350
Normalize expression 1 and separate it into component groups using the 'NormalizeExpressionAndSeparateGroups'. (Fig. 18B) Let 'expr1' be the array of individual characters for expression 1. Let 'ptrlist1' be the array of component group descriptor codes for expression 1.
Intersect Expressions
Normalize expression 2 and separate it into component groups using the 'NormalizeExpressionAndSeparateGroups'. (Fig. 18B) Let 'expr2' be the array of individual characters for expression 2. Let 'ptrlist2' be the array of component group descriptor codes for expression 2.
18K352
Let curisectptr = 0.
18K354
Compare the component groups of the expressions using the 'CompareComponentGroups', with cns1=false, p1=0, cns2=false, p2=0, and result='equal'.
18K356
If the expressions have extra data associated with them, compare it now. Merge the extra data comparison result with the expression comparison result using table 7.
18K358
18K366
Construct the intersection expression using the 'ConstructIntersections'. (Fig. 18N)
18K360
Is the merged comparison result 'disjoint' ?
No
Yes
18K367
If the expressions have extra data associated with them, intersect it now. Associate the intersected data with each subexpression of the intersection expression.
Return the intersection.

Figure 18K

Figure 18L


Note Consuming
18L242
Let cnsstart = p1.
18L244
Is swapped true ?
Yes
18L246
Add the length of ptrlist2 to cnsstart.
No
18L248
Has an intersection notation buffer been created ?
No
18L250
Create an intersecton notation buffer.
Yes
18L252
Does cnsstart match cnsstart of the last notation added ?
Yes
18L254
Move curisectptr to the start of the last notation.
No
18L256
In the notation buffer at index curisectptr, add a notation with the values of cnsstart, p2, and i2.
18L25?
Advance curisectptr.
Done

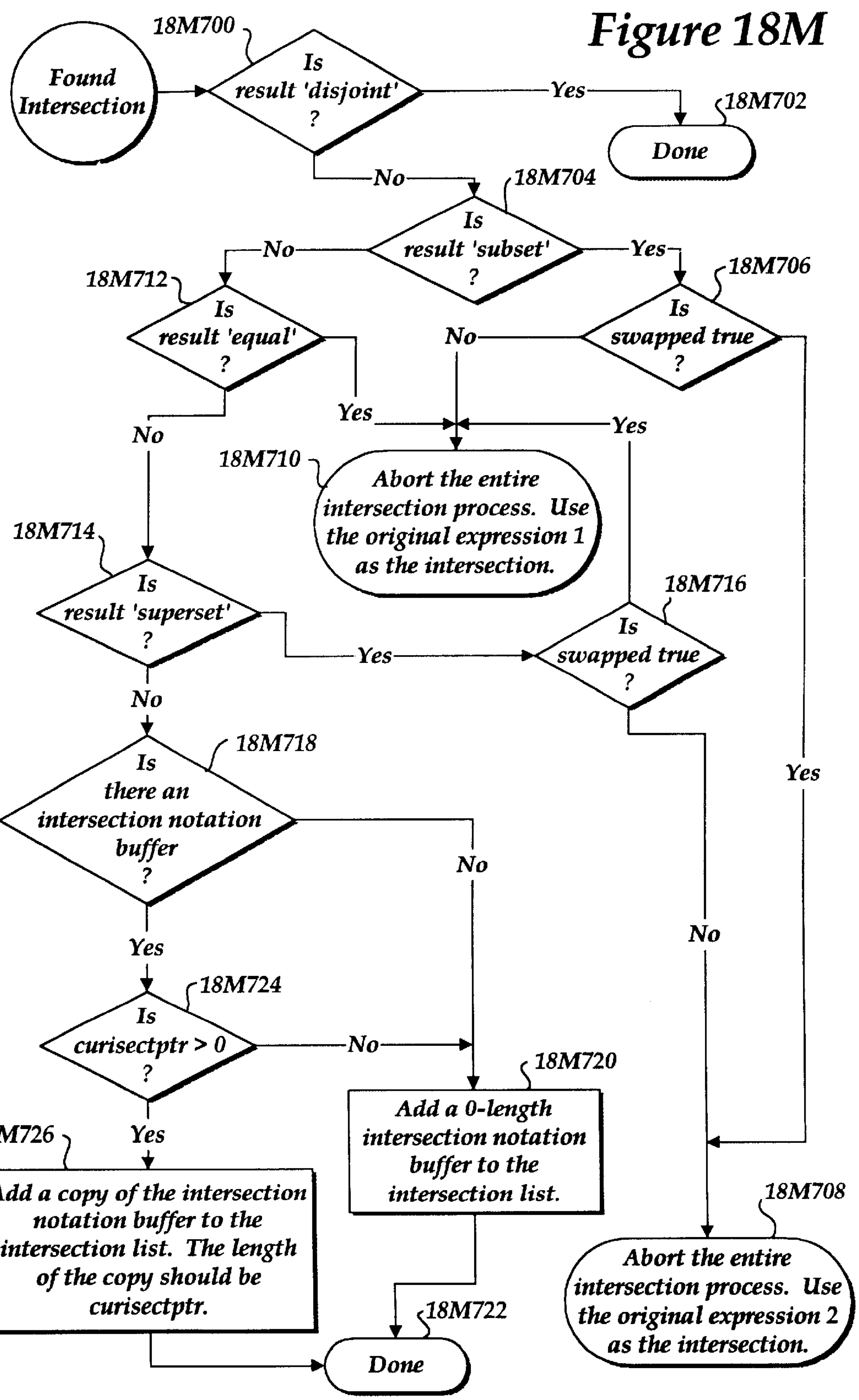


Figure 18M
Found Intersection
18M700
Is result 'disjoint' ?
Yes
18M702
Done
No
18M704
Is result 'subset' ?
No
Yes
18M712
Is result 'equal' ?
18M706
Is swapped true ?
No
Yes
Yes
No
18M710
Abort the entire intersection process. Use the original expression 1 as the intersection.
18M714
Is result 'superset' ?
18M716
Is swapped true ?
Yes
No
18M718
Is there an intersection notation buffer ?
No
Yes
No
Yes
18M724
Is curisectptr > 0 ?
No
18M720
Add a 0-length intersection notation buffer to the intersection list.
18M726
Yes
Add a copy of the intersection notation buffer to the intersection list. The length of the copy should be curisectptr.
18M708
Abort the entire intersection process. Use the original expression 2 as the intersection.
18M722
Done

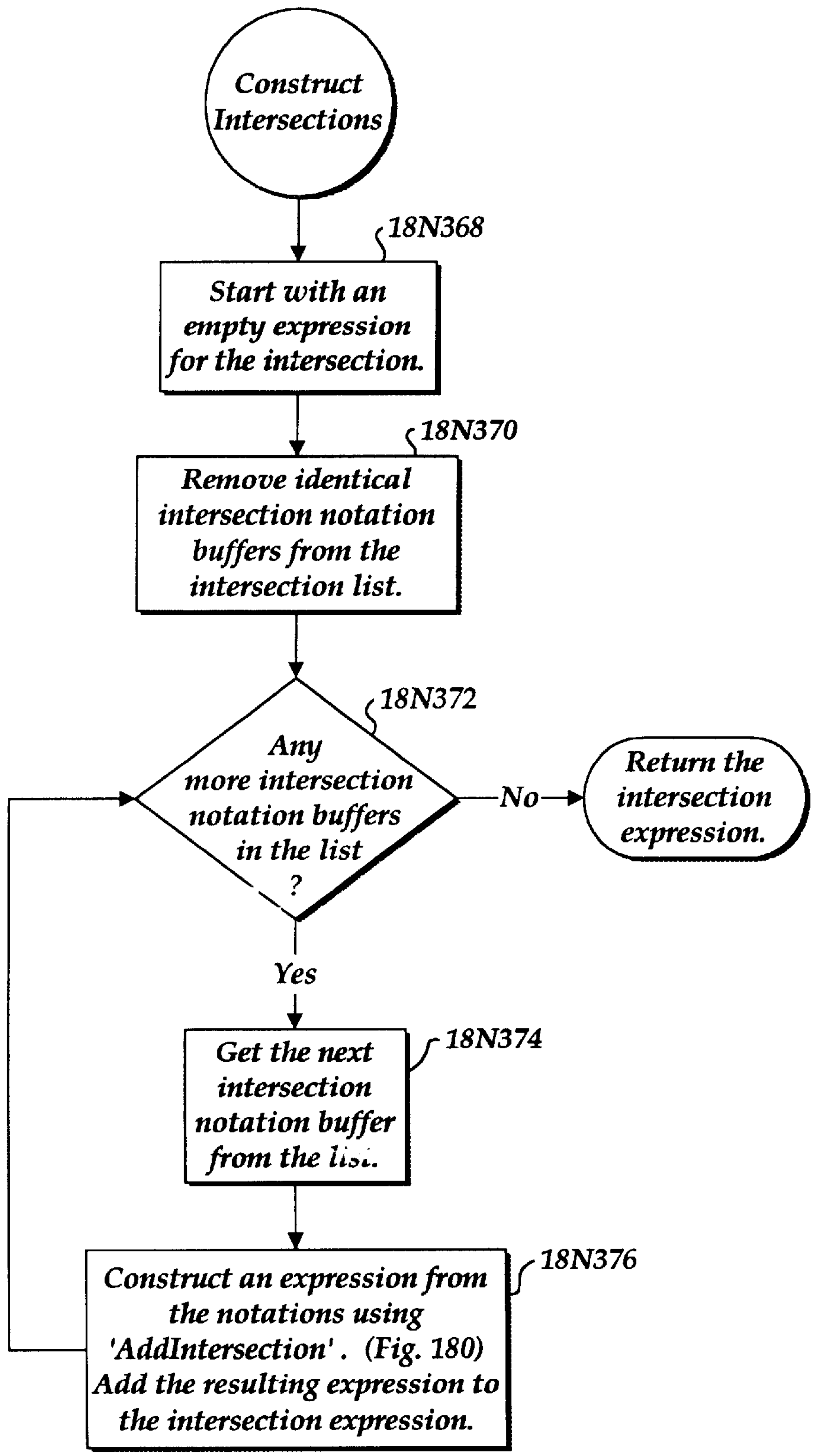
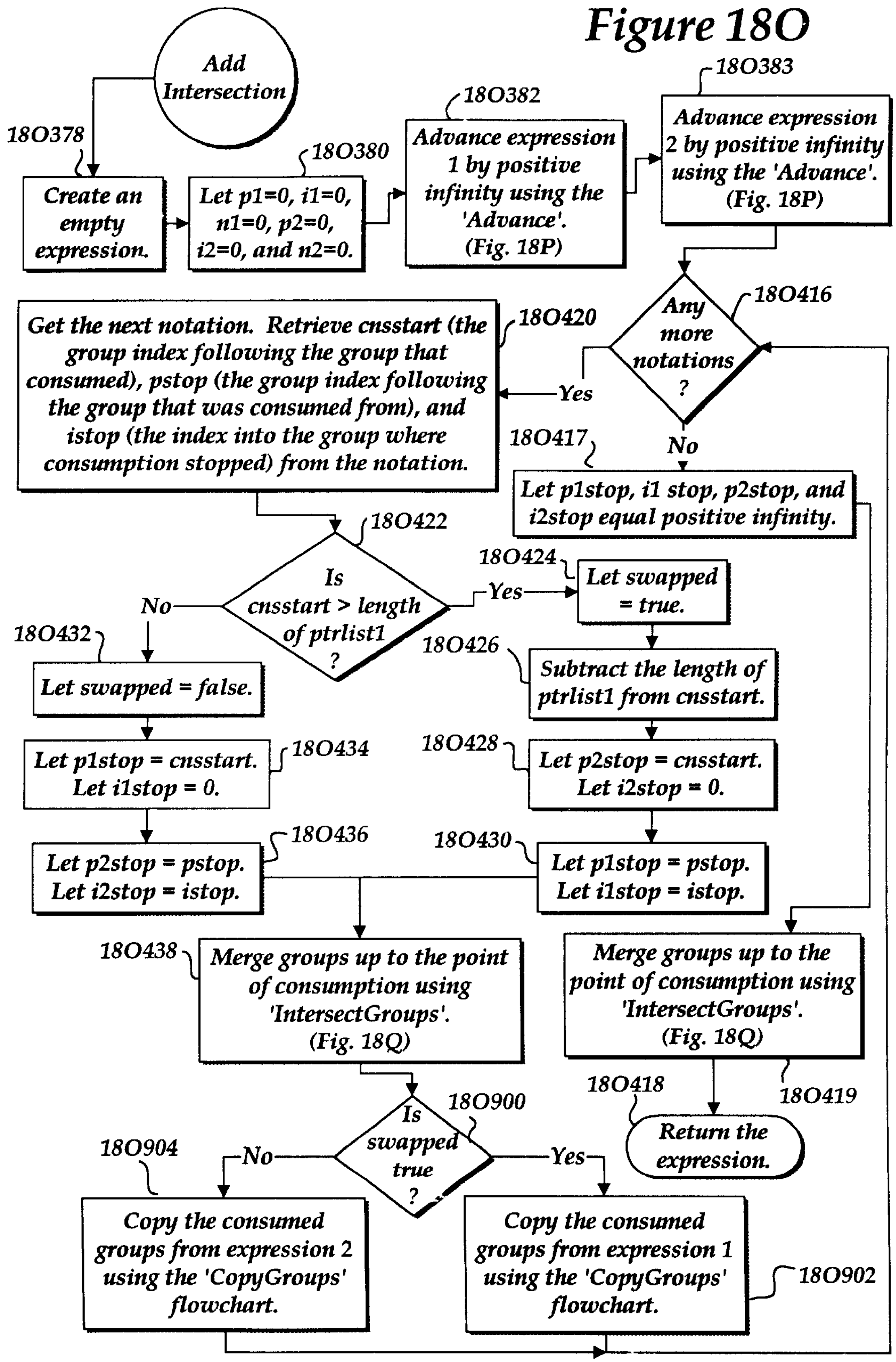

Construct Intersections
18N368
Start with an empty expression for the intersection.
18N370
Remove identical intersection notation buffers from the intersection list.
18N372
Any more intersection notation buffers in the list ?
No
Return the intersection expression.
Yes
18N374
Get the next intersection notation buffer from the list.
18N376
Construct an expression from the notations using 'AddIntersection'. (Fig. 180) Add the resulting expression to the intersection expression.

Figure 18N

Figure 18O


Add Intersection
18O378
Create an empty expression.
18O380
Let p1=0, i1=0, n1=0, p2=0, i2=0, and n2=0.
18O382
Advance expression 1 by positive infinity using the 'Advance'. (Fig. 18P)
18O383
Advance expression 2 by positive infinity using the 'Advance'. (Fig. 18P)
18O416
Any more notations ?
Yes
No
18O420
Get the next notation. Retrieve cnsstart (the group index following the group that consumed), pstop (the group index following the group that was consumed from), and istop (the index into the group where consumption stopped) from the notation.
18O417
Let p1stop, i1 stop, p2stop, and i2stop equal positive infinity.
18O422
Is cnsstart > length of ptrlist1 ?
No
Yes
18O424
Let swapped = true.
18O432
Let swapped = false.
18O426
Subtract the length of ptrlist1 from cnsstart.
18O434
Let p1stop = cnsstart. Let i1stop = 0.
18O428
Let p2stop = cnsstart. Let i2stop = 0.
18O436
Let p2stop = pstop. Let i2stop = istop.
18O430
Let p1stop = pstop. Let i1stop = istop.
18O438
Merge groups up to the point of consumption using 'IntersectGroups'. (Fig. 18Q)
Merge groups up to the point of consumption using 'IntersectGroups'. (Fig. 18Q)
18O418
18O419
Return the expression.
18O900
Is swapped true ?
No
Yes
18O904
Copy the consumed groups from expression 2 using the 'CopyGroups' flowchart.
Copy the consumed groups from expression 1 using the 'CopyGroups' flowchart.
18O902

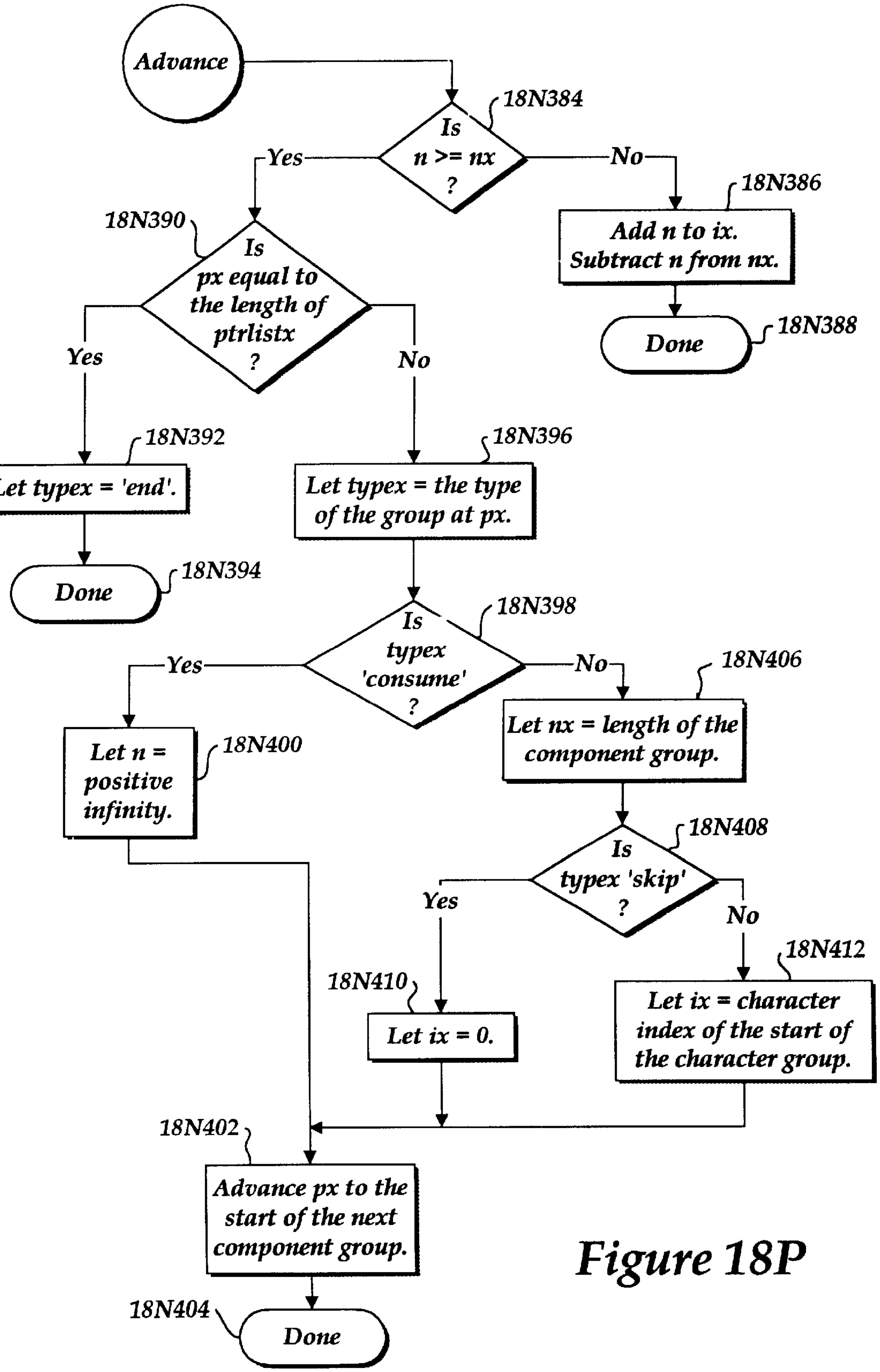

Advance
18N384
Is n >= nx ?
Yes
No
18N386
Add n to ix. Subtract n from nx.
Done
18N388
18N390
Is px equal to the length of ptrlistx ?
Yes
No
18N392
Let typex = 'end'.
Done
18N394
18N396
Let typex = the type of the group at px.
18N398
Is typex 'consume' ?
Yes
No
18N406
Let nx = length of the component group.
Let n = positive infinity.
18N400
18N408
Is typex 'skip' ?
Yes
No
18N412
18N410
Let ix = 0.
Let ix = character index of the start of the character group.
18N402
Advance px to the start of the next component group.
18N404
Done

Figure 18P

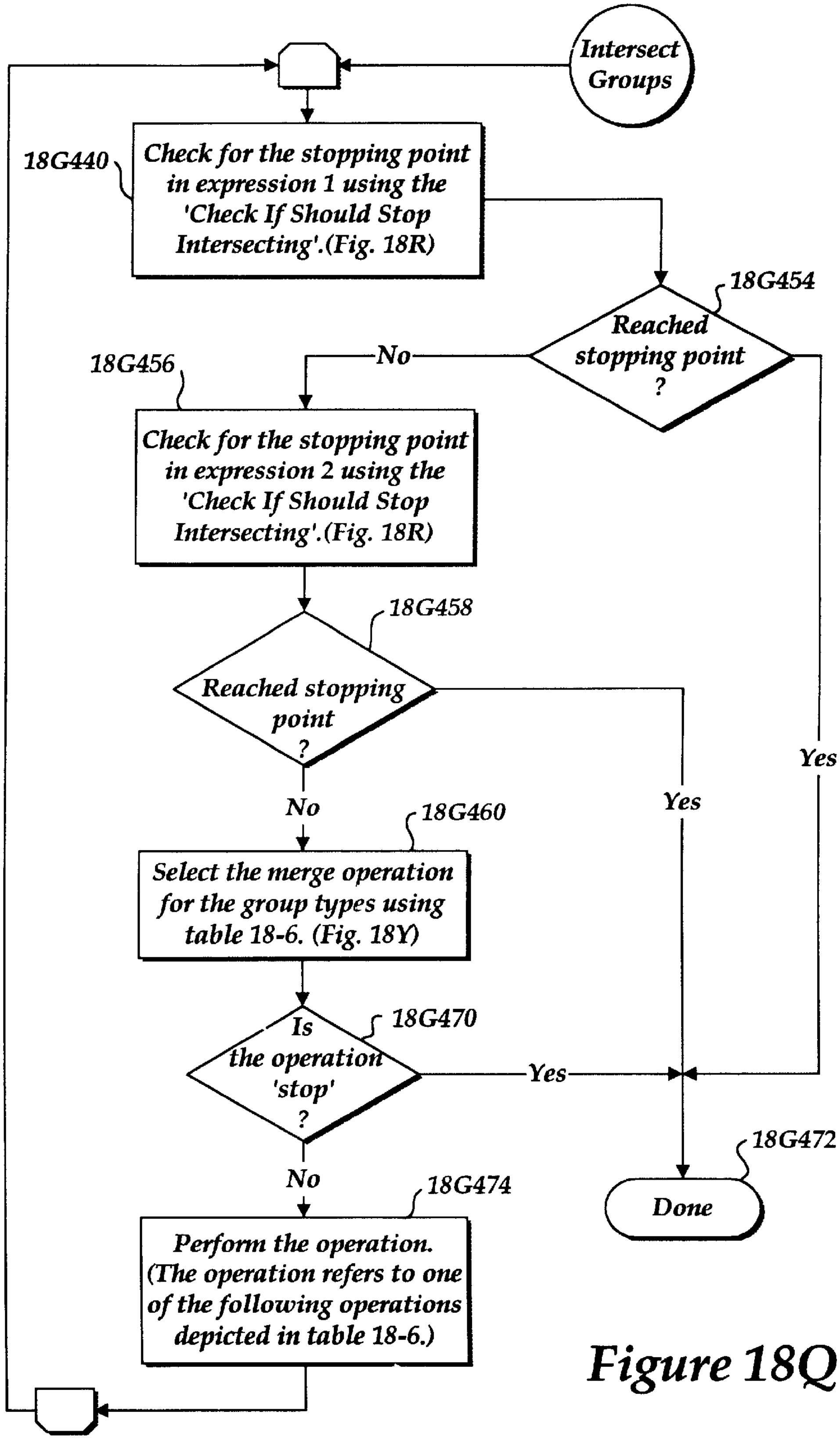

Intersect Groups
18G440
Check for the stopping point in expression 1 using the 'Check If Should Stop Intersecting'.(Fig. 18R)
18G454
Reached stopping point ?
No
Yes
18G456
Check for the stopping point in expression 2 using the 'Check If Should Stop Intersecting'.(Fig. 18R)
18G458
Reached stopping point ?
Yes
No
18G460
Select the merge operation for the group types using table 18-6. (Fig. 18Y)
18G470
Is the operation 'stop' ?
Yes
No
18G472
Done
18G474
Perform the operation. (The operation refers to one of the following operations depicted in table 18-6.)

Figure 18Q

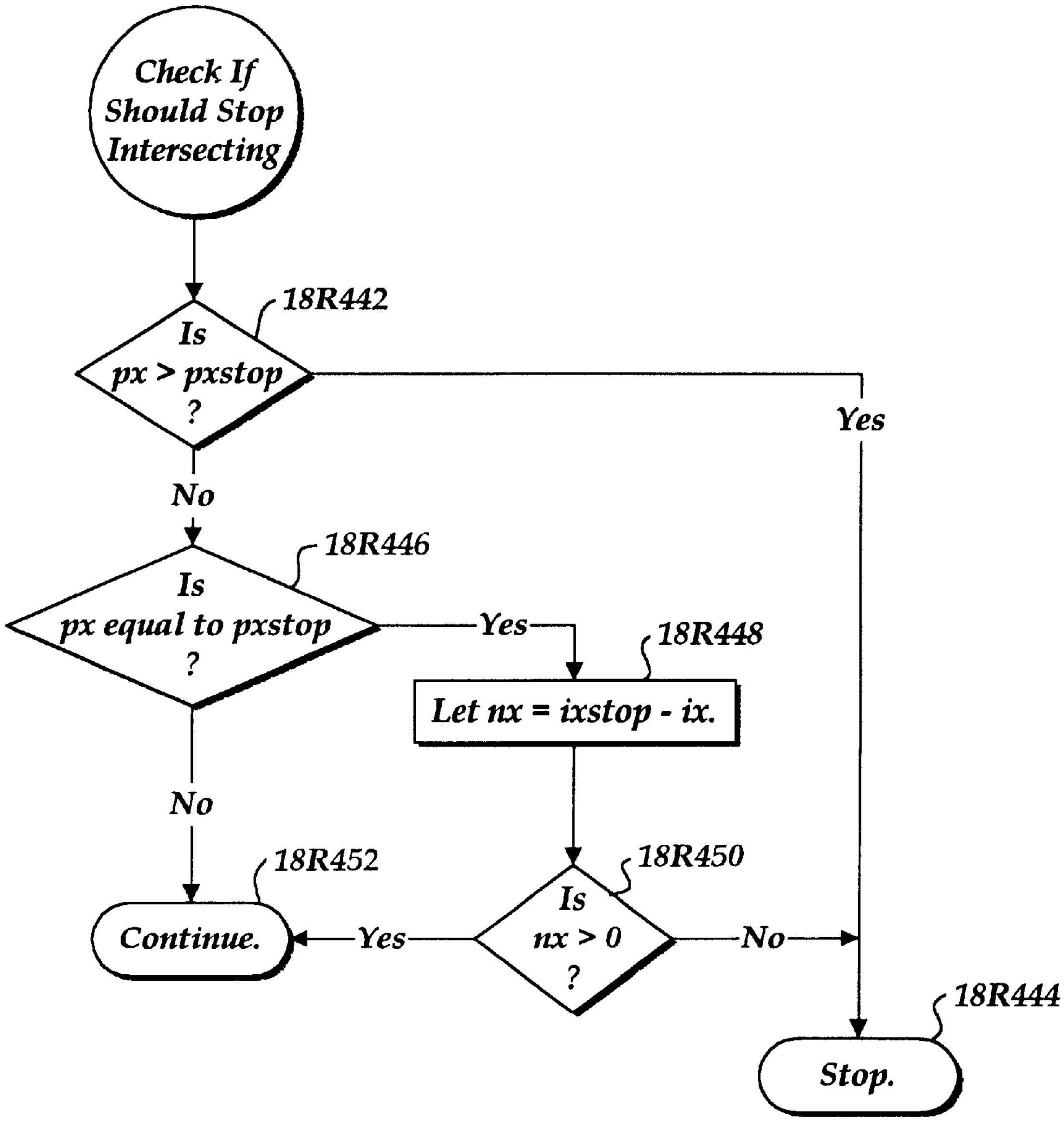

Check If Should Stop Intersecting
18R442
Is px > pxstop ?
Yes
No
18R446
Is px equal to pxstop ?
Yes
18R448
Let nx = ixstop - ix.
No
18R452
18R450
Continue.
Yes
Is nx > 0 ?
No
18R444
Stop.

Figure 18R

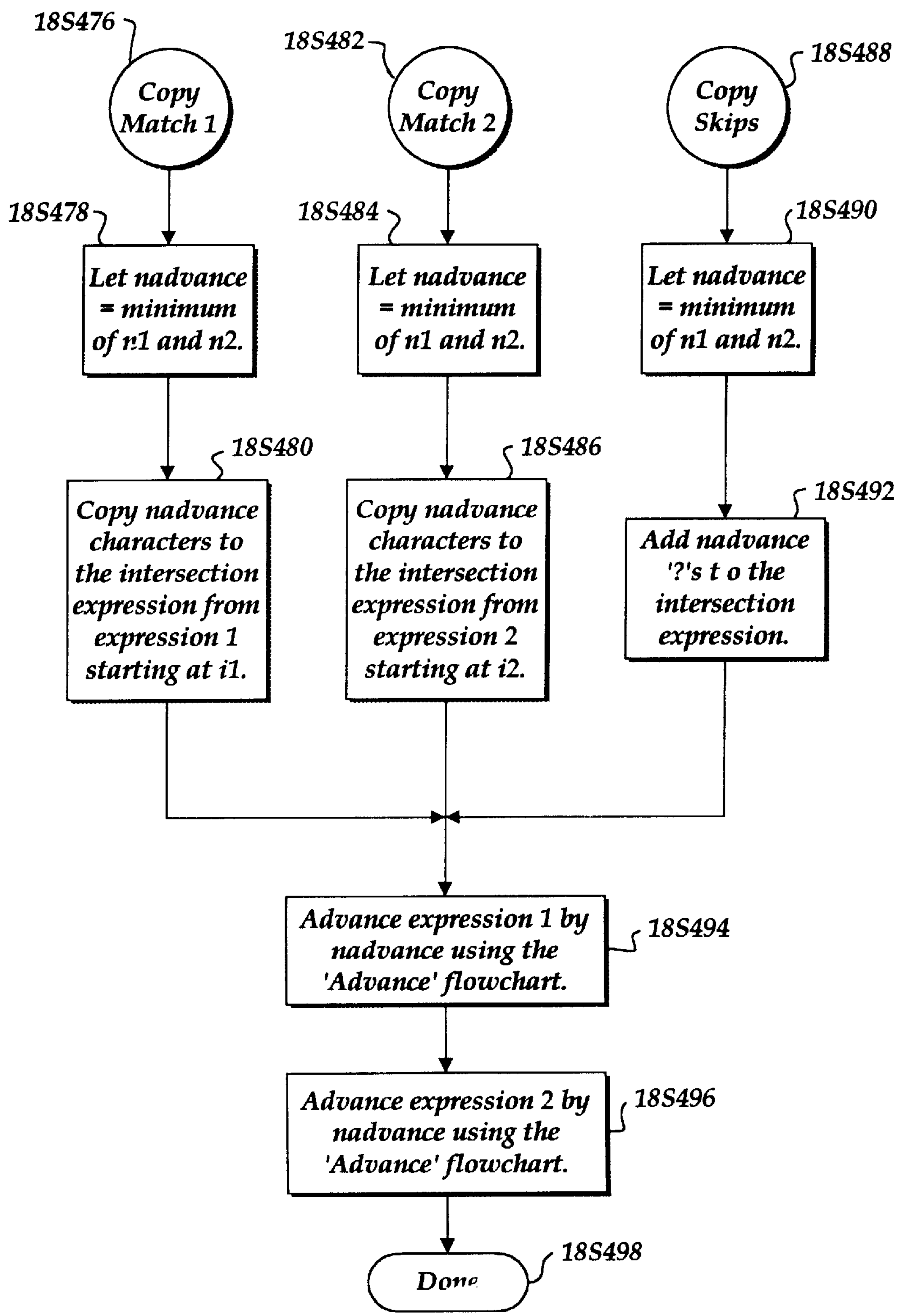

18S476
Copy Match 1
18S482
Copy Match 2
18S488
Copy Skips
18S478
Let nadvance = minimum of n1 and n2.
18S484
Let nadvance = minimum of n1 and n2.
18S490
Let nadvance = minimum of n1 and n2.
18S480
Copy nadvance characters to the intersection expression from expression 1 starting at i1.
18S486
Copy nadvance characters to the intersection expression from expression 2 starting at i2.
18S492
Add nadvance '?'s t o the intersection expression.
Advance expression 1 by nadvance using the 'Advance' flowchart.
18S494
Advance expression 2 by nadvance using the 'Advance' flowchart.
18S496
Done
18S498

Figure 18S

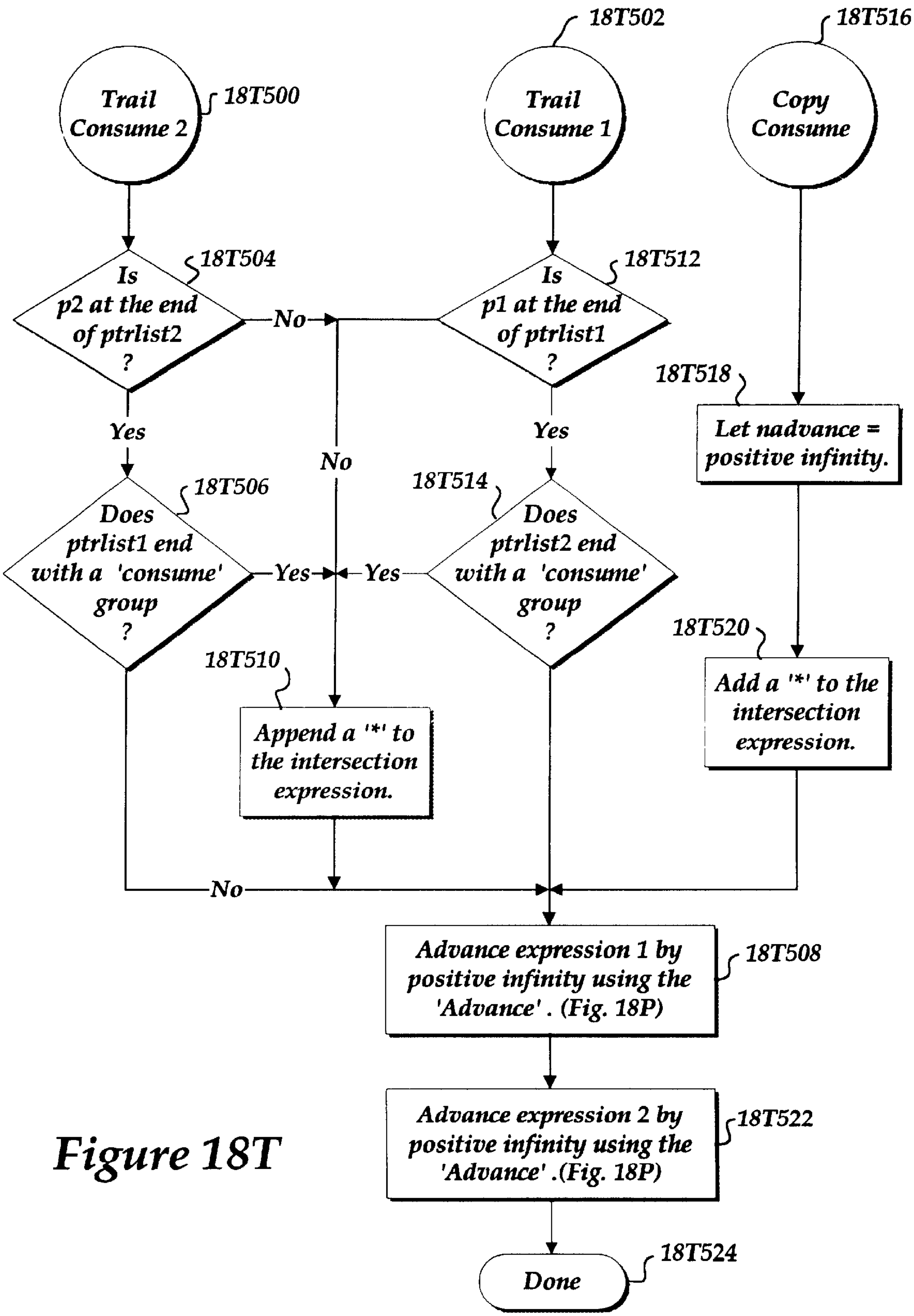

18T500
Trail Consume 2
18T502
Trail Consume 1
18T516
Copy Consume
18T504
Is p2 at the end of ptrlist2 ?
No
18T512
Is p1 at the end of ptrlist1 ?
Yes
No
Yes
18T518
Let nadvance = positive infinity.
18T506
Does ptrlist1 end with a 'consume' group ?
Yes
Yes
18T514
Does ptrlist2 end with a 'consume' group ?
18T520
Add a '*' to the intersection expression.
18T510
Append a '*' to the intersection expression.
No
Advance expression 1 by positive infinity using the 'Advance' . (Fig. 18P)
18T508
Advance expression 2 by positive infinity using the 'Advance' .(Fig. 18P)
18T522
Done
18T524

Figure 18T

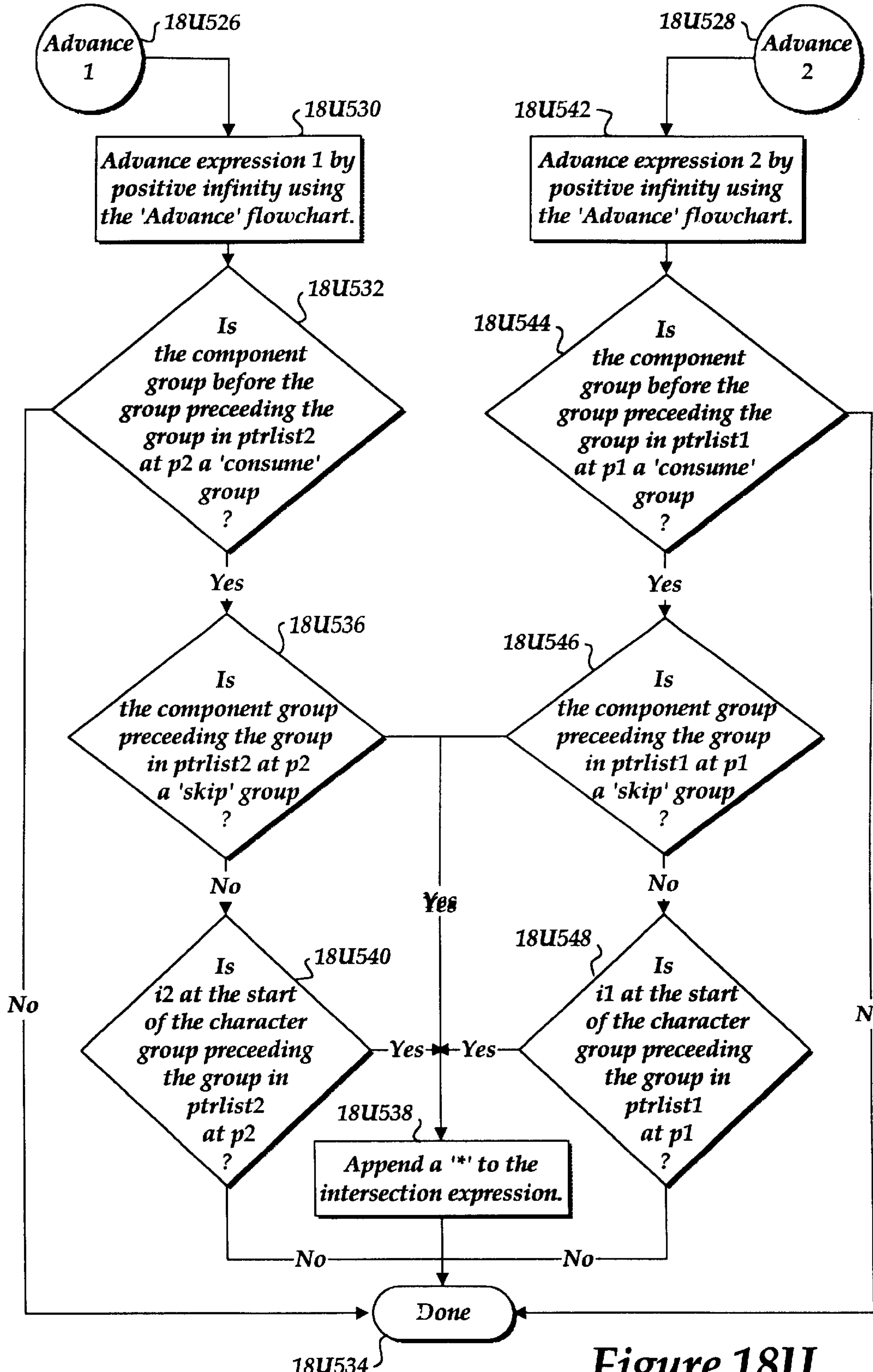
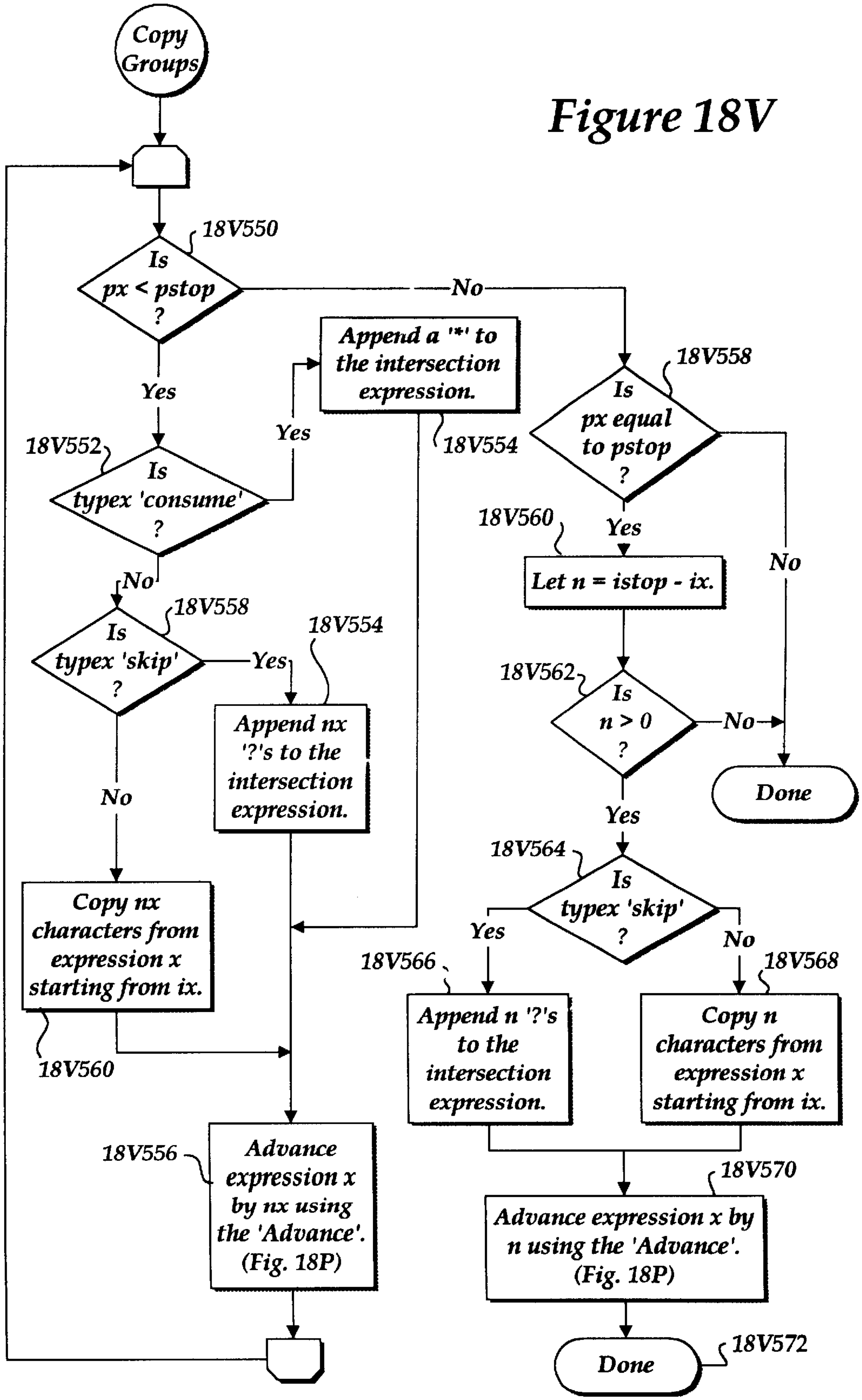

Advance 1
18U526
Advance 2
18U528
18U530
Advance expression 1 by positive infinity using the 'Advance' flowchart.
18U542
Advance expression 2 by positive infinity using the 'Advance' flowchart.
18U532
Is the component group before the group preceeding the group in ptrlist2 at p2 a 'consume' group ?
18U544
Is the component group before the group preceeding the group in ptrlist1 at p1 a 'consume' group ?
Yes
Yes
18U536
Is the component group preceeding the group in ptrlist2 at p2 a 'skip' group ?
18U546
Is the component group preceeding the group in ptrlist1 at p1 a 'skip' group ?
No
Yes
No
18U540
Is i2 at the start of the character group preceeding the group in ptrlist2 at p2 ?
18U548
Is i1 at the start of the character group preceeding the group in ptrlist1 at p1 ?
No
No
Yes
Yes
18U538
Append a '*' to the intersection expression.
No
No
Done
18U534

Figure 18U

Figure 18V


Copy Groups
18V550
Is px < pstop ?
No
Yes
Append a '*' to the intersection expression.
18V554
18V558
Is px equal to pstop ?
18V552
Is typex 'consume' ?
Yes
No
18V558
Is typex 'skip' ?
Yes
18V554
Append nx '?'s to the intersection expression.
No
Copy nx characters from expression x starting from ix.
18V560
18V560
Yes
Let n = istop - ix.
No
18V562
Is n > 0 ?
No
Done
Yes
18V564
Is typex 'skip' ?
Yes
No
18V566
18V568
Append n '?'s to the intersection expression.
Copy n characters from expression x starting from ix.
18V556
Advance expression x by nx using the 'Advance'. (Fig. 18P)
18V570
Advance expression x by n using the 'Advance'. (Fig. 18P)
Done
18V572

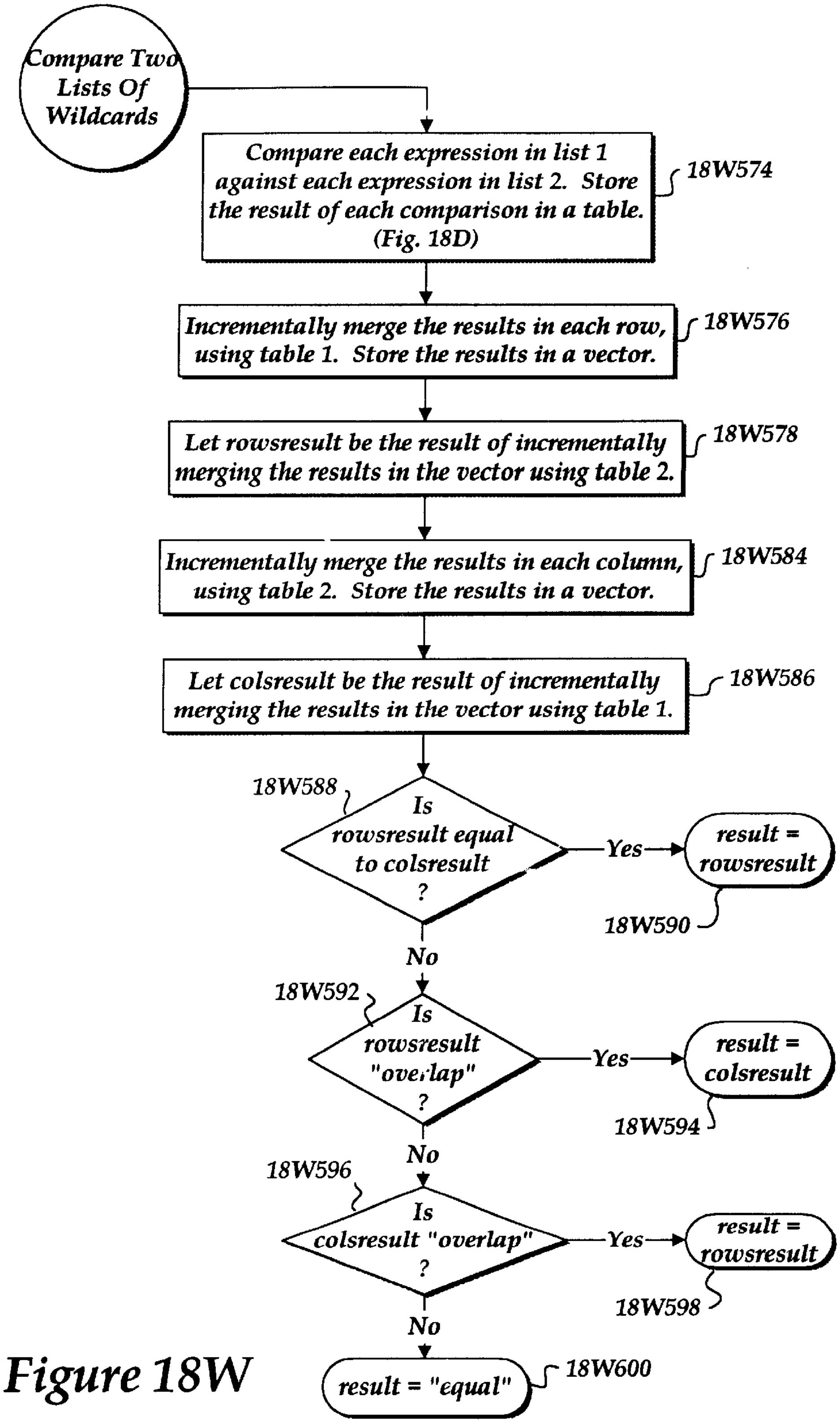

Compare Two Lists Of Wildcards
Compare each expression in list 1 against each expression in list 2. Store the result of each comparison in a table. (Fig. 18D)
18W574
Incrementally merge the results in each row, using table 1. Store the results in a vector.
18W576
Let rowsresult be the result of incrementally merging the results in the vector using table 2.
18W578
Incrementally merge the results in each column, using table 2. Store the results in a vector.
18W584
Let colsresult be the result of incrementally merging the results in the vector using table 1.
18W586
18W588
Is rowsresult equal to colsresult ?
Yes
result = rowsresult
18W590
No
18W592
Is rowsresult "overlap" ?
Yes
result = colsresult
18W594
No
18W596
Is colsresult "overlap" ?
Yes
result = rowsresult
18W598
No
result = "equal"
18W600

Figure 18W

TABLE 18-1 – ROW RESULT MERGING

	Overlap	Disjoint	Subset	Equal	Superset
Overlap	Overlap	Overlap	Subset	Subset	Overlap
Disjoint	Overlap	Disjoint	Subset	Subset	Overlap
Subset	Subset	Subset	Subset	Subset	Subset
Equal	Subset	Subset	Subset	Equal	Equal
Superset	Overlap	Overlap	Subset	Equal	Superset

TABLE 18-2 – COLUMN RESULT MERGING

	Overlap	Disjoint	Subset	Equal	Superset
Overlap	Overlap	Overlap	Overlap	Superset	Superset
Disjoint	Overlap	Disjoint	Overlap	Superset	Superset
Subset	Overlap	Overlap	Subset	Equal	Superset
Equal	Superset	Superset	Equal	Equal	Superset
Superset	Superset	Superset	Superset	Superset	Superset

TABLE 18-3 – ADJUSTMENT FOR CONSUMPTION FROM TARGET EXPRESSION

Original result	Biased result
Overlap	Overlap
Disjoint	Disjoint
Subset	Overlap
Equal	Superset
Superset	Superset

TABLE 18-4 – ADJUSTMENT FOR CONSUMPTION FROM SOURCE EXPRESSION

Original result	Biased result
Overlap	Overlap
Disjoint	Disjoint
Subset	Subset
Equal	Subset
Superset	Overlap

Figure 18X

TABLE 18-5

Original result	Inverted result
Overlap	Overlap
Disjoint	Disjoint
Subset	Superset
Equal	Equal
Superset	Subset

TABLE 18-6 – FLOWCHART TO USE FOR MERGING GROUPS FROM THE EXPRESSIONS.

	Match	Skip	Consume	End
Match	CopyMatch1	CopyMatch1	Advance2	CopyMatch1
Skip	CopyMatch2	CopySkips	Advance2	CopySkips
Consume	Advance1	Advance1	CopyConsume	TrailConsume1
End	CopyMatch2	CopySkips	TrailConsume2	(Stop merging)

TABLE 18-7 – MERGING "EXTRA DATA" COMPARISON RESULTS WITH EXPRESSION COMPARISON RESULTS.

18Y316

	Empty	Empty subset	Empty superset	Overlap	Disjoint	Subset	Equal	Superset
Overlap	Overlap	Overlap	Overlap	Overlap	Disjoint	Overlap	Overlap	Overlap
Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint	Disjoint
Subset	Subset	Subset	Disjoint	Overlap	Disjoint	Subset	Subset	Disjoint
Equal	Equal	Subset	Superset	Overlap	Disjoint	Subset	Equal	Superset
Superset	Superset	Disjoint	Superset	Overlap	Disjoint	Disjoint	Superset	Superset

18Y314

18Y318

Figure 18Y

TABLE 18-8 SAMPLE EXPRESSION COMPARISONS AND INTERSECTIONS.

Expression 1	Expression 2	Relationship	Intersection
*?	*?	equal	
*?	?	superset	
*??	??	superset	
?*?	??	superset	
??*?	???	superset	
??*??	????	superset	
?*?*?	???	superset	
??	??	superset	
?	?	superset	
???*	???	superset	
*a?a	*a?*a	subset	
?*a*	b*a*	superset	
?a	b*a*	superset	
?*?a	a?*?*?	overlap	a?*?a
??*a	a??*	overlap	a*?a
?*a	a??*a	superset	
?*a	a?a*	overlap	a?a;a?a*a
??*a	a???*	overlap	a*??a
?a*	a??*	overlap	aa?*
?*	a??	superset	
??*	a??	superset	
abc	*?	subset	
?*a	*?	subset	
*a?	*b	overlap	*ab
a*a*	a*?*	subset	
a?	*?	subset	
a*?	*?	subset	
*ba	?a	overlap	ba
?	a	superset	
?	?	equal	
??	?	disjoint	
???	???	equal	
a	??	disjoint	
a	?a	disjoint	
ab	a?	subset	
abc	a??	subset	
*a	???a	superset	
*a	???ba	superset	
???ab	x*b	overlap	x??ab
???a	x*a	overlap	x??a
*a	a?aa*	overlap	a?aa;a?aa*a
*a	aa?a*	overlap	aa?a;aa?a*a

Figure 18Z

*a	?a*?	overlap	?a*a
*a	?a?*	overlap	?a*a
*a	??	overlap	?a
*a	??*	overlap	?*a
a	??*	overlap	*a?*;*?a*
a	?*	subset	
a	?	overlap	a
a	a	superset	
a*?a*	a*a*?a*	superset	
a*?a*	*a*a*	subset	
a*?a*	*?a*	subset	
a*?a*	a*aa*	superset	
a*?a*	?a*	overlap	aa*a*
a	a	equal	
a	b	disjoint	
ab	a	disjoint	
a*	a*	equal	
a*	a	superset	
""	""	equal	
?	""	disjoint	
""	*	subset	
a*	""	disjoint	
a	""	disjoint	
*a	a	superset	
*a	*a	equal	
*a	a*	overlap	a;a*a
*a	ba	superset	
*a	bcdefga	superset	
a*c	*c	subset	
a*b*c	abc	superset	
a*b*c	a*b*c	equal	
a*d*c	a*c	subset	
*abcd	abcd	superset	
*abcd	abcde	disjoint	
*abcd	abcd*	overlap	abcd;abcd*abcd
*a*b	*ab	superset	
*a*b	*cab	superset	
*abc*d	*bcd	overlap	*abcd;*abc*bcd
abc*d	*bcd	overlap	abcd;abc*bcd
*ab*cd	*abc*d	overlap	*abcd;*abc*cd
ab*c	ab*	subset	
a*bc	ab*c	overlap	abc;ab*bc
*bc	abc*	overlap	abc*bc;abc
*abc	bc*	overlap	bc*abc
a*bc	ab*bc	superset	
*a*b	*ab	superset	
*a*a*	*aa*	superset	
a*b*c	a*d*c	overlap	a*b*d*c;a*d*b*c
*a?a	*a?*a	subset	
a*	*a*	subset	

Figure 18AA

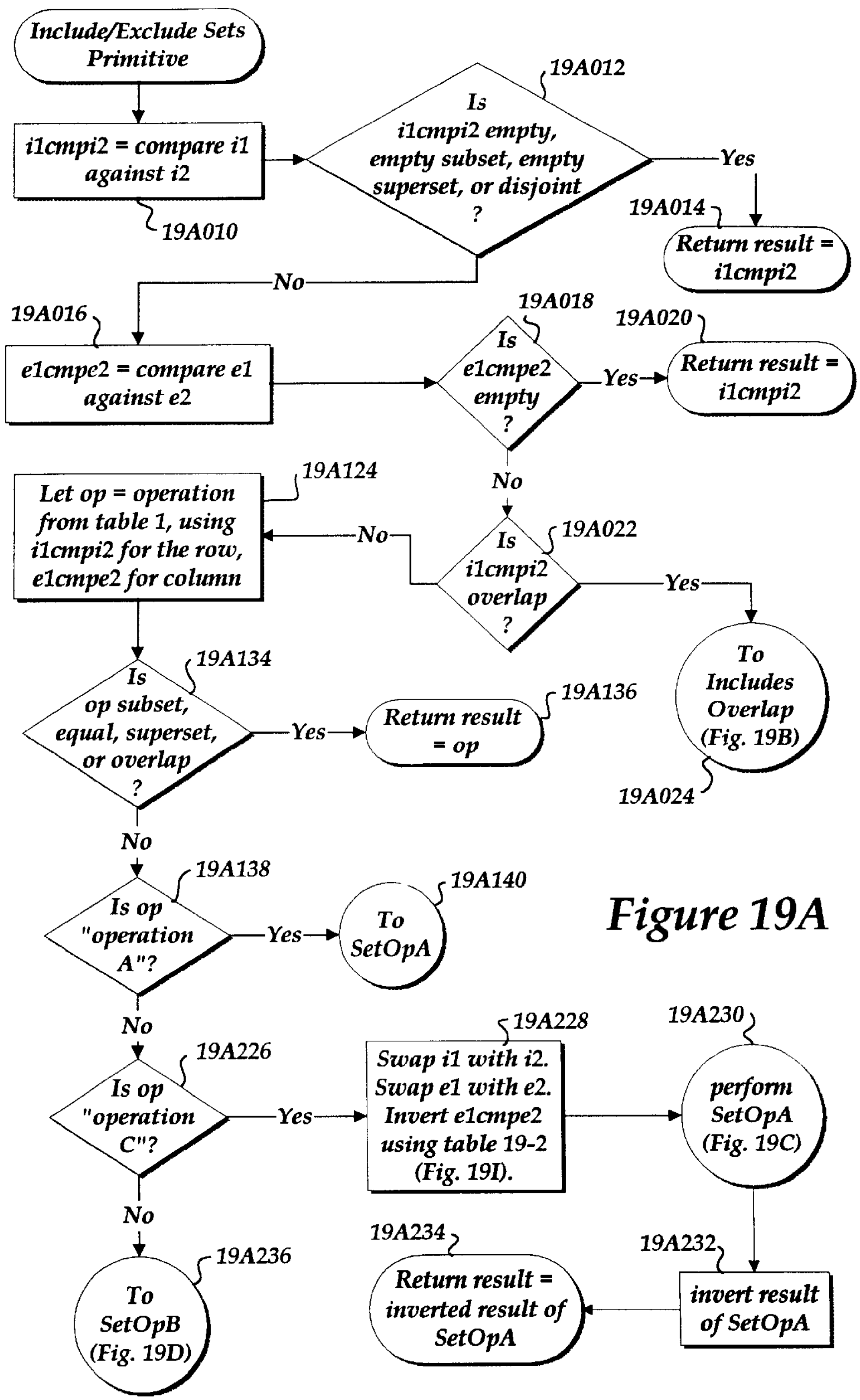

Include/Exclude Sets Primitive
i1cmpi2 = compare i1 against i2
19A010
Is i1cmpi2 empty, empty subset, empty superset, or disjoint ?
19A012
Yes
19A014
Return result = i1cmpi2
No
19A016
e1cmpe2 = compare e1 against e2
19A018
Is e1cmpe2 empty ?
Yes
19A020
Return result = i1cmpi2
No
19A022
Is i1cmpi2 overlap ?
No
Yes
19A124
Let op = operation from table 1, using i1cmpi2 for the row, e1cmpe2 for column
To Includes Overlap (Fig. 19B)
19A024
19A134
Is op subset, equal, superset, or overlap ?
Yes
Return result = op
19A136
No
19A138
Is op "operation A"?
Yes
To SetOpA
19A140
No
19A226
Is op "operation C"?
Yes
19A228
Swap i1 with i2. Swap e1 with e2. Invert e1cmpe2 using table 19-2 (Fig. 19I).
19A230
perform SetOpA (Fig. 19C)
No
19A236
To SetOpB (Fig. 19D)
19A234
Return result = inverted result of SetOpA
19A232
invert result of SetOpA

Figure 19A

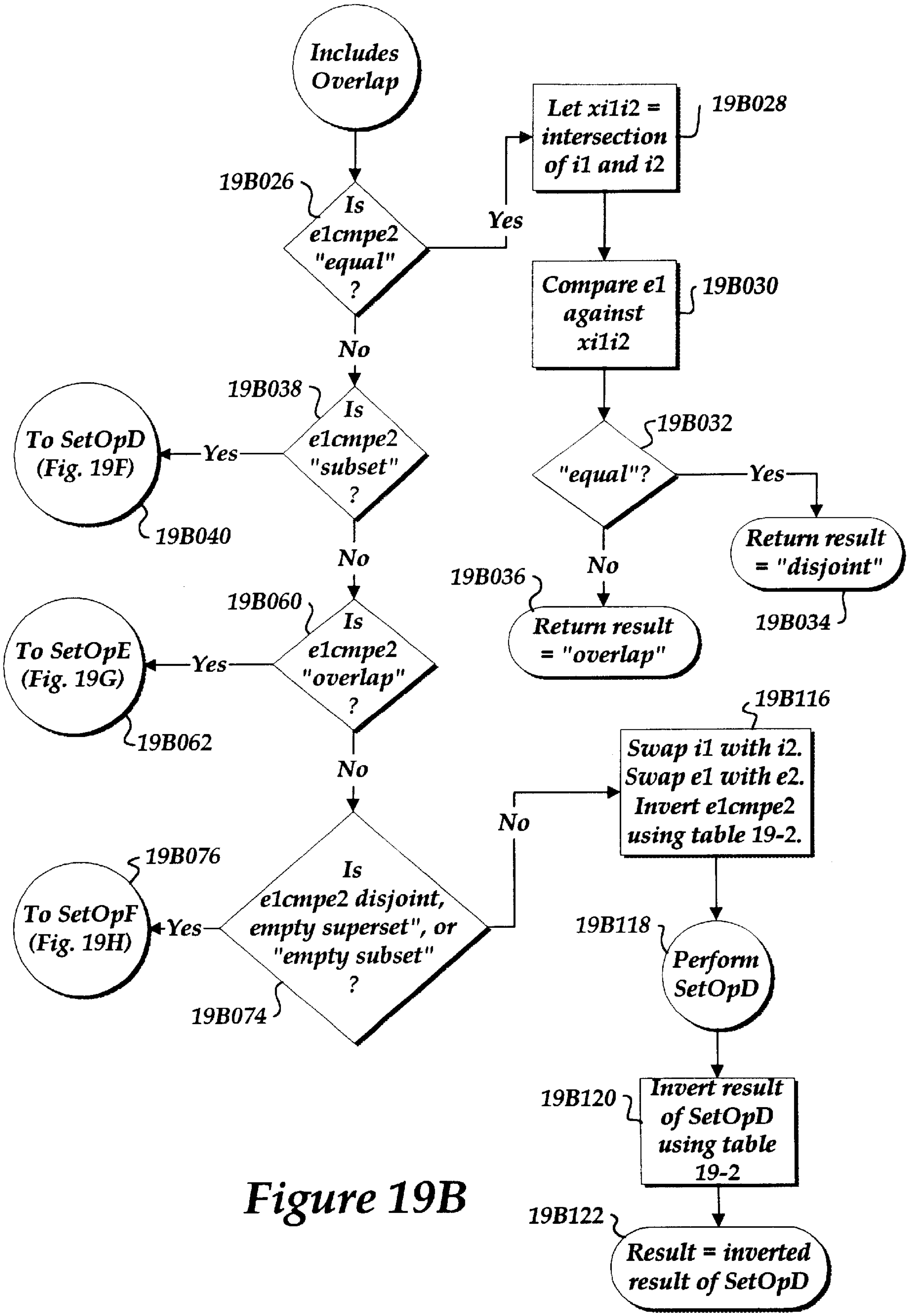
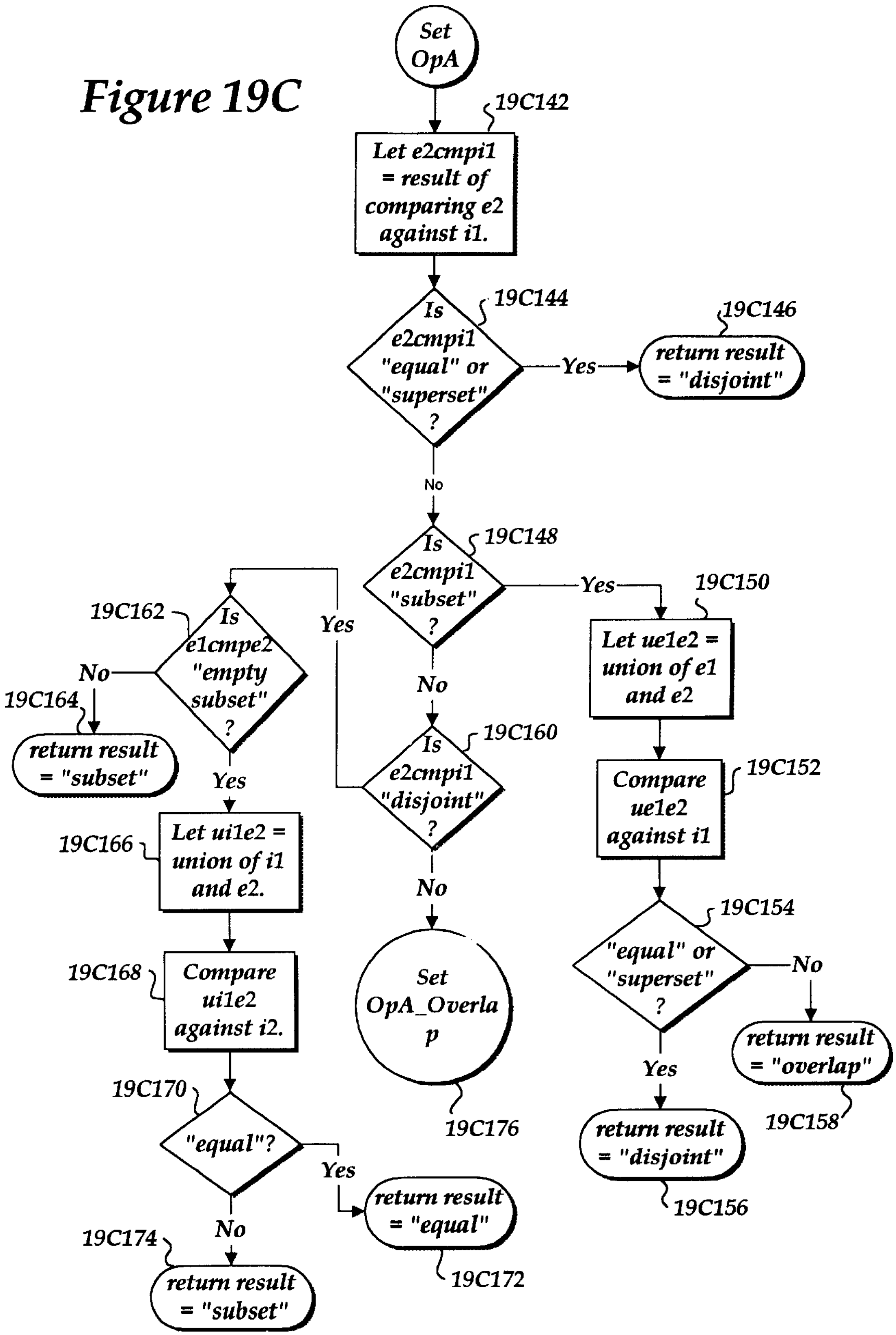

Includes Overlap
19B026
Is e1cmpe2 "equal" ?
Yes
No
Let xi1i2 = intersection of i1 and i2
19B028
Compare e1 against xi1i2
19B030
19B032
"equal"?
Yes
No
Return result = "disjoint"
19B034
19B036
Return result = "overlap"
19B038
Is e1cmpe2 "subset" ?
Yes
To SetOpD (Fig. 19F)
19B040
No
19B060
Is e1cmpe2 "overlap" ?
Yes
To SetOpE (Fig. 19G)
19B062
No
Is e1cmpe2 disjoint, empty superset", or "empty subset" ?
19B074
Yes
To SetOpF (Fig. 19H)
19B076
No
19B116
Swap i1 with i2. Swap e1 with e2. Invert e1cmpe2 using table 19-2.
19B118
Perform SetOpD
19B120
Invert result of SetOpD using table 19-2
19B122
Result = inverted result of SetOpD

Figure 19B

Figure 19C


Set OpA
19C142
Let e2cmpi1 = result of comparing e2 against i1.
19C144
Is e2cmpi1 "equal" or "superset" ?
Yes
19C146
return result = "disjoint"
No
19C148
Is e2cmpi1 "subset" ?
Yes
19C150
Let ue1e2 = union of e1 and e2
19C152
Compare ue1e2 against i1
19C154
"equal" or "superset" ?
No
return result = "overlap"
19C158
Yes
return result = "disjoint"
19C156
No
19C160
Is e2cmpi1 "disjoint" ?
Yes
No
Set OpA_Overlap
19C176
19C162
Is e1cmpe2 "empty subset" ?
No
19C164
return result = "subset"
Yes
19C166
Let ui1e2 = union of i1 and e2.
19C168
Compare ui1e2 against i2.
19C170
"equal"?
Yes
return result = "equal"
19C172
No
19C174
return result = "subset"

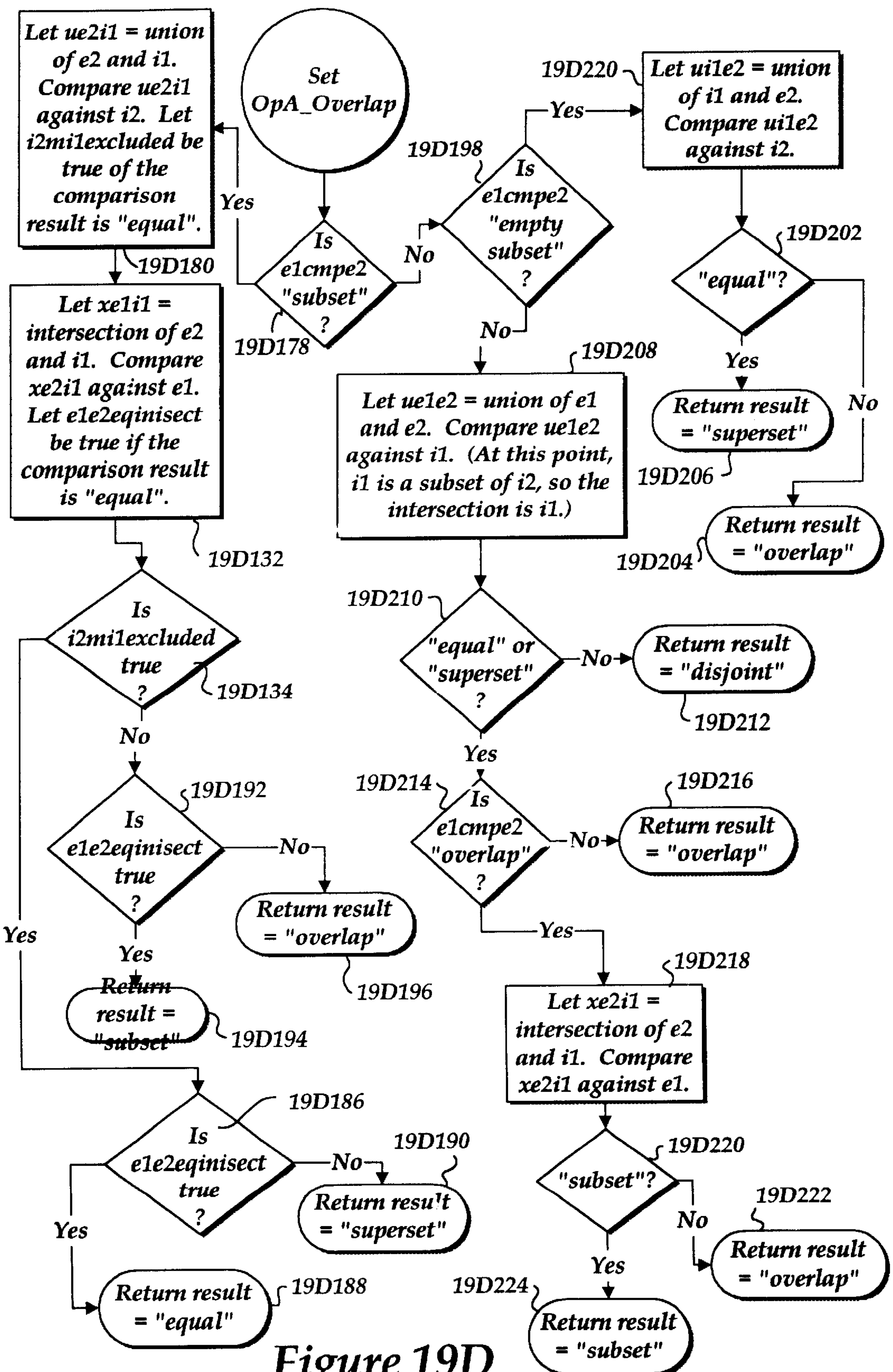

Set OpA_Overlap
Is e1cmpe2 "subset" ?
19D178
Yes
Let ue2i1 = union of e2 and i1. Compare ue2i1 against i2. Let i2mi1excluded be true of the comparison result is "equal".
19D180
Let xe1i1 = intersection of e2 and i1. Compare xe2i1 against e1. Let e1e2eqinisect be true if the comparison result is "equal".
19D132
Is i2mi1excluded true ?
19D134
No
Is e1e2eqinisect true ?
19D192
No
Return result = "overlap"
19D196
Yes
Return result = "subset"
19D194
Yes
Is e1e2eqinisect true ?
19D186
No
Return result = "superset"
19D190
Yes
Return result = "equal"
19D188
No
Is e1cmpe2 "empty subset" ?
19D198
Yes
Let ui1e2 = union of i1 and e2. Compare ui1e2 against i2.
19D220
"equal"?
19D202
Yes
Return result = "superset"
19D206
No
Return result = "overlap"
19D204
No
Let ue1e2 = union of e1 and e2. Compare ue1e2 against i1. (At this point, i1 is a subset of i2, so the intersection is i1.)
19D208
"equal" or "superset" ?
19D210
No
Return result = "disjoint"
19D212
Yes
Is e1cmpe2 "overlap" ?
19D214
No
Return result = "overlap"
19D216
Yes
Let xe2i1 = intersection of e2 and i1. Compare xe2i1 against e1.
19D218
"subset"?
19D220
No
Return result = "overlap"
19D222
Yes
Return result = "subset"
19D224

Figure 19D

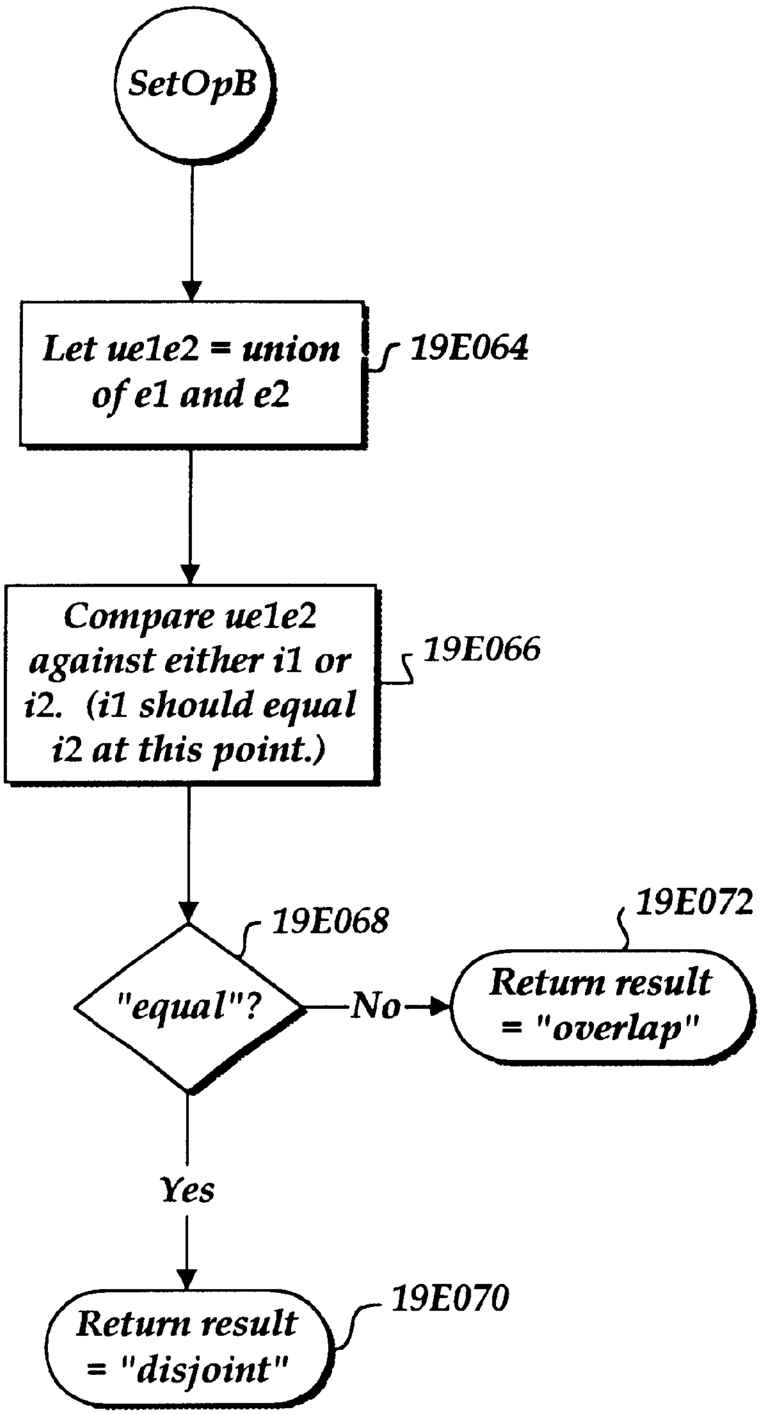

SetOpB
Let ue1e2 = union of e1 and e2
19E064
Compare ue1e2 against either i1 or i2. (i1 should equal i2 at this point.)
19E066
19E068
"equal"?
No
Return result = "overlap"
19E072
Yes
Return result = "disjoint"
19E070

Figure 19E

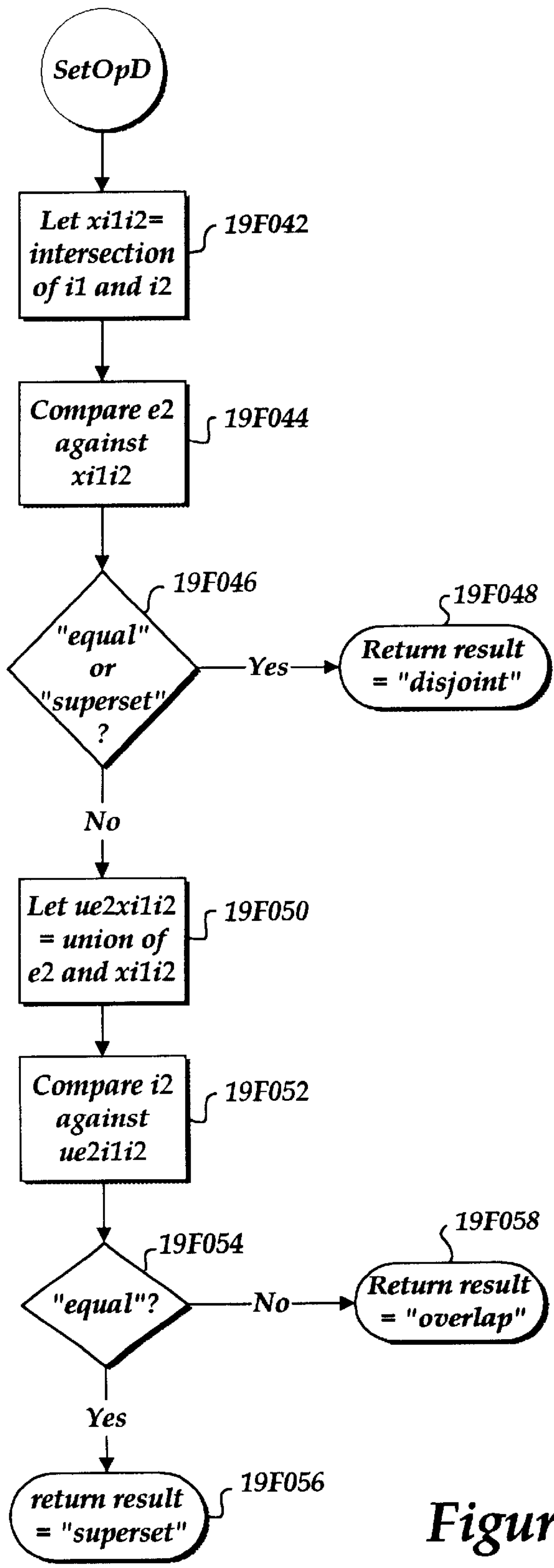

SetOpD
Let xi1i2= intersection of i1 and i2
19F042
Compare e2 against xi1i2
19F044
"equal" or "superset" ?
19F046
Yes
Return result = "disjoint"
19F048
No
Let ue2xi1i2 = union of e2 and xi1i2
19F050
Compare i2 against ue2i1i2
19F052
"equal"?
19F054
No
Return result = "overlap"
19F058
Yes
return result = "superset"
19F056

Figure 19F

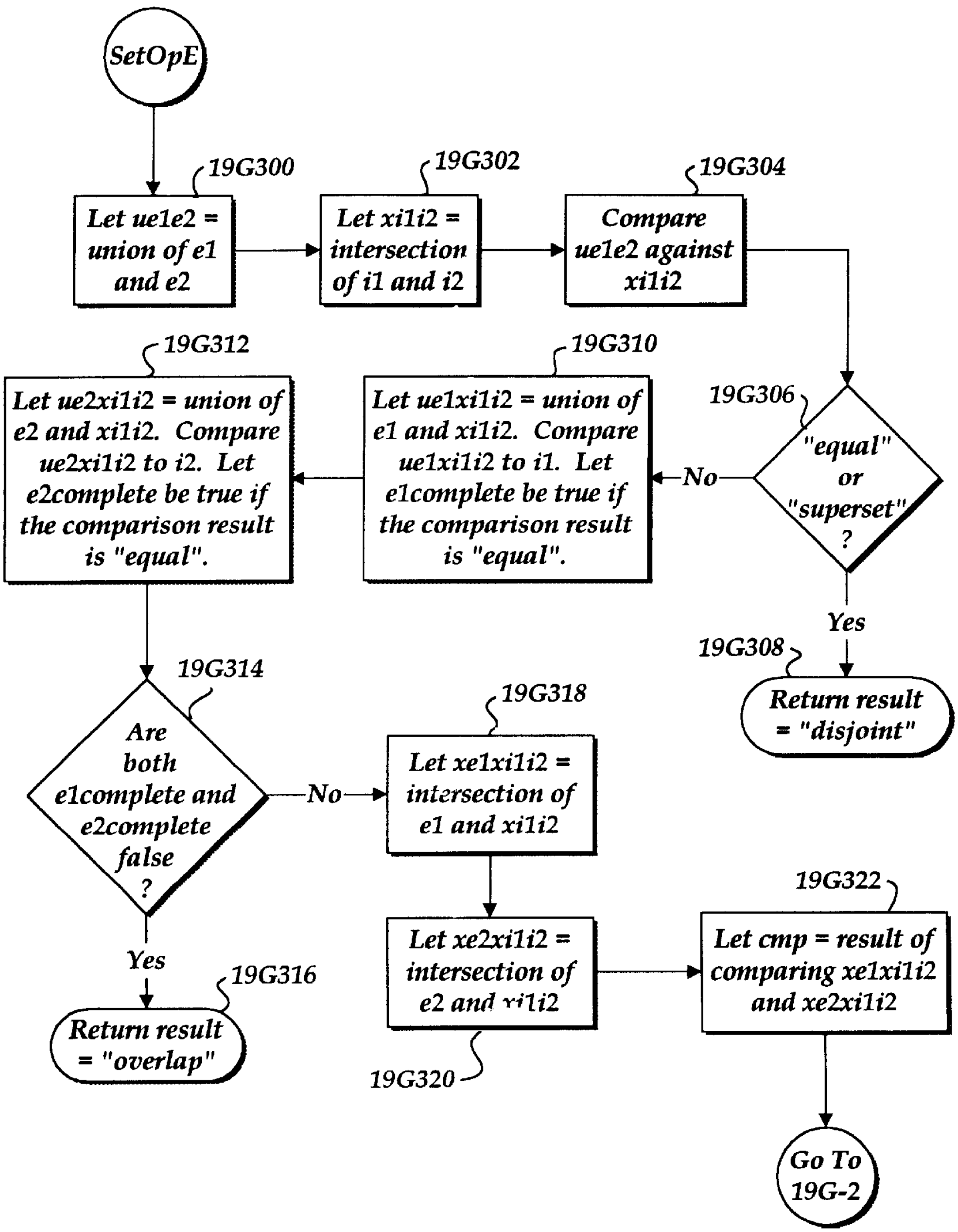
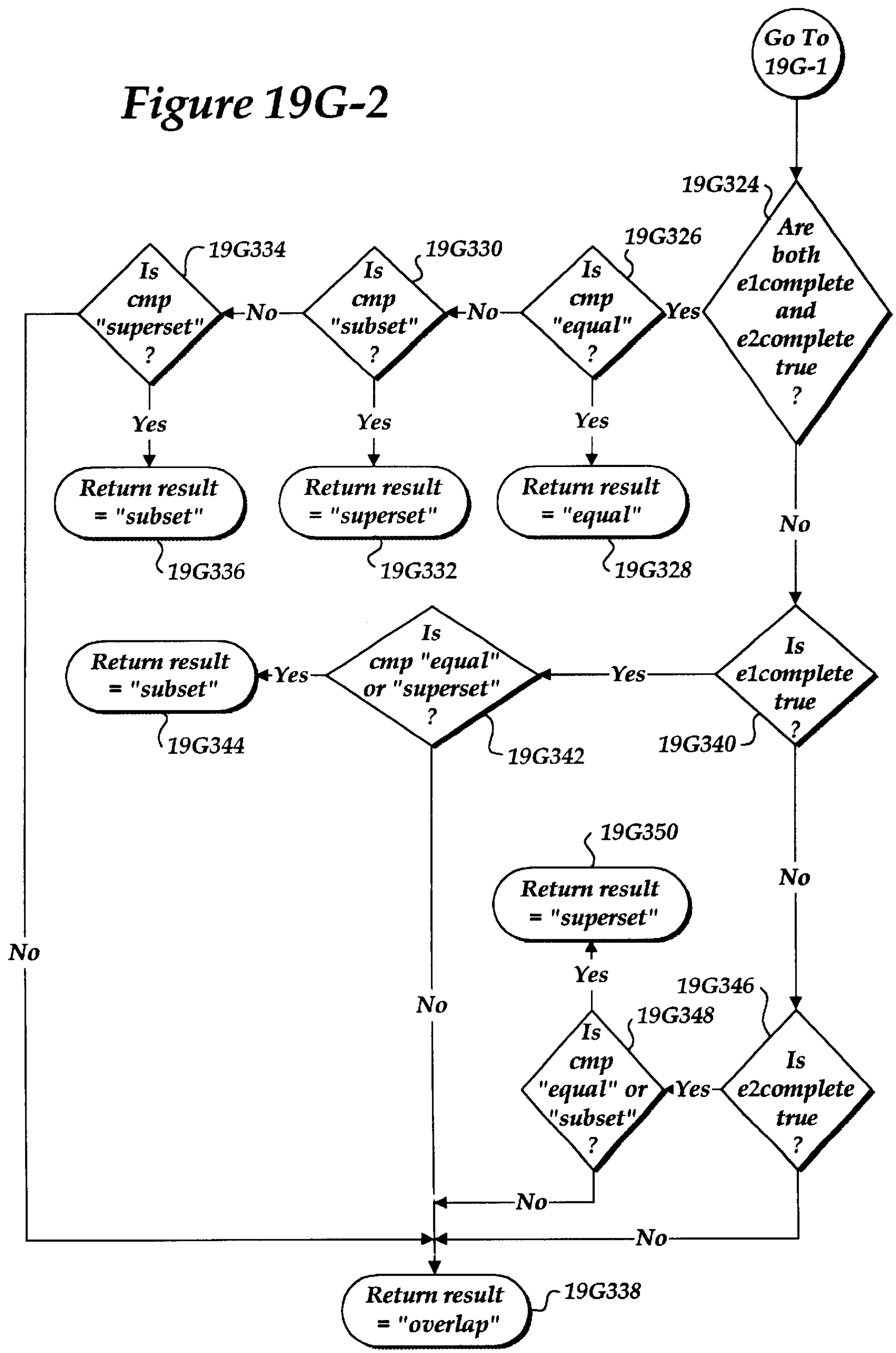

SetOpE
19G300
Let ue1e2 = union of e1 and e2
19G302
Let xi1i2 = intersection of i1 and i2
19G304
Compare ue1e2 against xi1i2
19G306
"equal" or "superset" ?
No
Yes
19G308
Return result = "disjoint"
19G310
Let ue1xi1i2 = union of e1 and xi1i2. Compare ue1xi1i2 to i1. Let e1complete be true if the comparison result is "equal".
19G312
Let ue2xi1i2 = union of e2 and xi1i2. Compare ue2xi1i2 to i2. Let e2complete be true if the comparison result is "equal".
19G314
Are both e1complete and e2complete false ?
Yes
19G316
Return result = "overlap"
No
19G318
Let xe1xi1i2 = intersection of e1 and xi1i2
19G320
Let xe2xi1i2 = intersection of e2 and xi1i2
19G322
Let cmp = result of comparing xe1xi1i2 and xe2xi1i2
Go To 19G-2

Figure 19G-1

Figure 19G-2


Go To 19G-1
19G324
Are both e1complete and e2complete true ?
Yes
No
19G326
Is cmp "equal" ?
Yes
Return result = "equal"
19G328
No
19G330
Is cmp "subset" ?
Yes
Return result = "superset"
19G332
No
19G334
Is cmp "superset" ?
Yes
Return result = "subset"
19G336
No
19G340
Is e1complete true ?
Yes
19G342
Is cmp "equal" or "superset" ?
Yes
Return result = "subset"
19G344
No
No
19G346
Is e2complete true ?
Yes
19G348
Is cmp "equal" or "subset" ?
Yes
Return result = "superset"
19G350
No
No
Return result = "overlap"
19G338

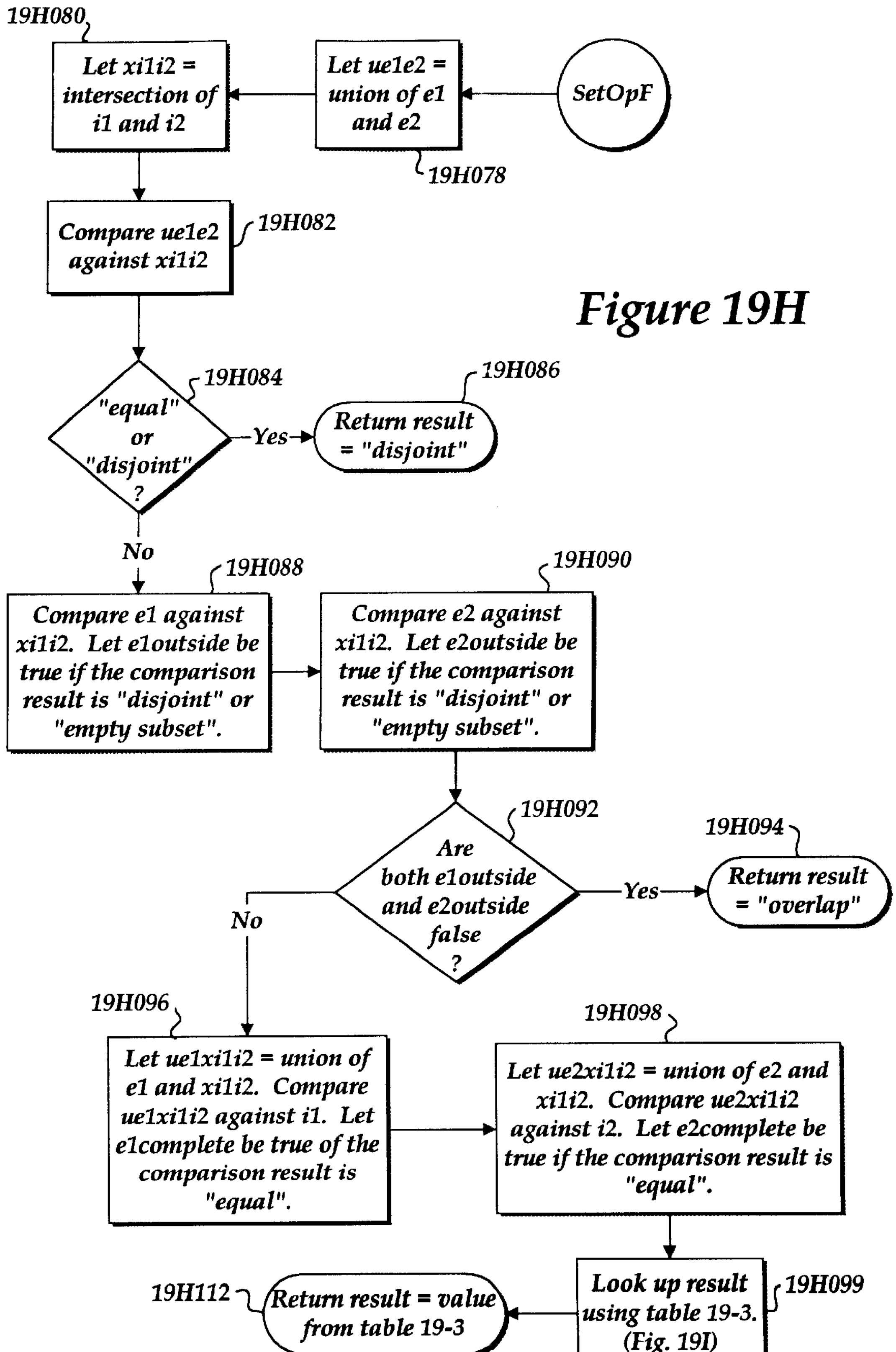

19H080
Let xi1i2 = intersection of i1 and i2
Let ue1e2 = union of e1 and e2
SetOpF
19H078
Compare ue1e2 against xi1i2
19H082
"equal" or "disjoint" ?
19H084
Yes
Return result = "disjoint"
19H086
No
19H088
Compare e1 against xi1i2. Let e1outside be true if the comparison result is "disjoint" or "empty subset".
19H090
Compare e2 against xi1i2. Let e2outside be true if the comparison result is "disjoint" or "empty subset".
19H092
Are both e1outside and e2outside false ?
Yes
19H094
Return result = "overlap"
No
19H096
Let ue1xi1i2 = union of e1 and xi1i2. Compare ue1xi1i2 against i1. Let e1complete be true of the comparison result is "equal".
19H098
Let ue2xi1i2 = union of e2 and xi1i2. Compare ue2xi1i2 against i2. Let e2complete be true if the comparison result is "equal".
Look up result using table 19-3. (Fig. 19I)
19H099
19H112
Return result = value from table 19-3

Figure 19H

Tables associated with figures for subtraction sets

19I130

Table 19-1

	Overlap	Disjoint	Subset	Equal	Superset	Empty Subset	Empty Superset
Subset	Operation A	Operation A	Operation A	Subset	Subset	Operation A	Subset
Equal	Operation C	Operation B	Superset	Equal	Subset	Superset	Subset
Superset	Operation C	Operation C	Superset	Superset	Superset	Superset	Operation C

19I128 19I132

Table 19-2

Original Comparison Result	Inverted Comparison Result
Superset	Subset
Subset	Superset
Empty Superset	Empty Subset
Empty Subset	Empty Superset
Equal	Equal
Disjoint	Disjoint
Empty	Empty
Overlap	Overlap

19I114

Figure 19I

19I106 19I108

Table 19-3

		E2outside	False	False	True	True
		E2complete	False	True	False	True
E1outside	E1complete					
False	False		Overlap	Overlap	Overlap	Overlap
False	True		Overlap	Overlap	Subset	Subset
True	False		Overlap	Superset	Overlap	Superset
True	True		Overlap	Superset	Subset	Equal

19I104 19I102 19I110

PREVIOUS/ACCUMULATED MERGE RESULTS	NEW MERGE RESULTS							
	1	2	3	4	5	6	7	8
A		OVERLAP	DISJOINT	SUBSET	EQUAL	SUPERSET	EMPTY SUBSET	EMPTY
B	EMPTY	OVERLAP	DISJOINT	SUBSET	EQUAL	SUPERSET	EMPTY SUBSET	EMPTY
C	EMPTY SUBSET	OVERLAP	DISJOINT	SUBSET	SUBSET	OVERLAP	DISJOINT	EMPTY SUBSET
D	EMPTY SUPERSET	OVERLAP	DISJOINT	OVERLAP	SUPERSET	SUPERSET	EMPTY SUBSET	EMPTY SUPERSET
E	DISJOINT	OVERLAP	DISJOINT	OVERLAP	OVERLAP	OVERLAP	DISJOINT	DISJOINT
F	OVERLAP	OVERLAP	OVERLAP	OVERLAP	OVERLAP	OVERLAP	OVERLAP	OVERLAP
G	SUBSET	OVERLAP	OVERLAP	SUBSET	SUBSET	OVERLAP	OVERLAP	SUBSET
H	SUPERSET	OVERLAP	OVERLAP	OVERLAP	SUPERSET	SUPERSET	SUPERSET	SUPERSET
I	EQUAL	OVERLAP	OVERLAP	SUBSET	EQUAL	SUPERSET	SUPERSET	EQUAL

FIGURE 20

COMPARE PERMISSIONS

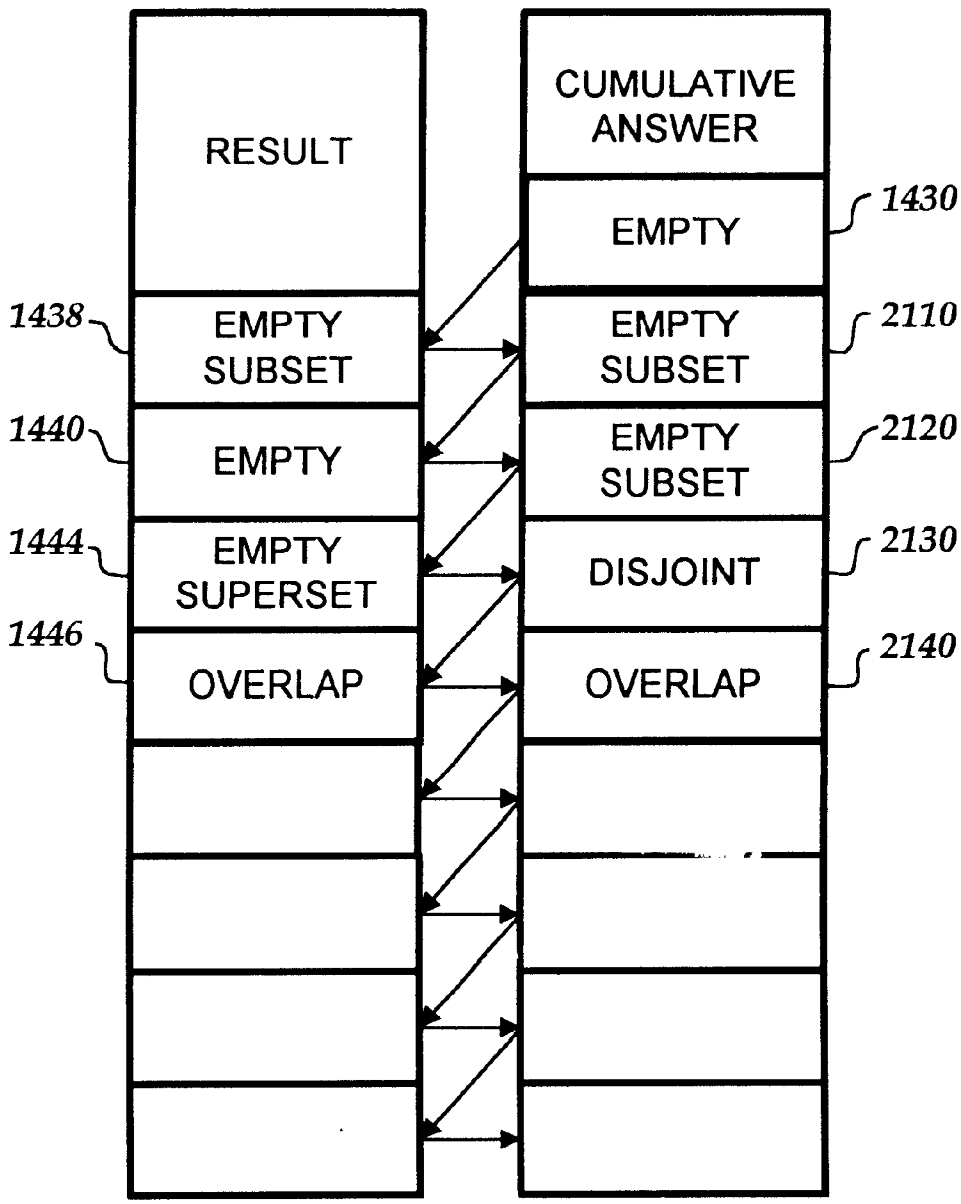


Figure 21

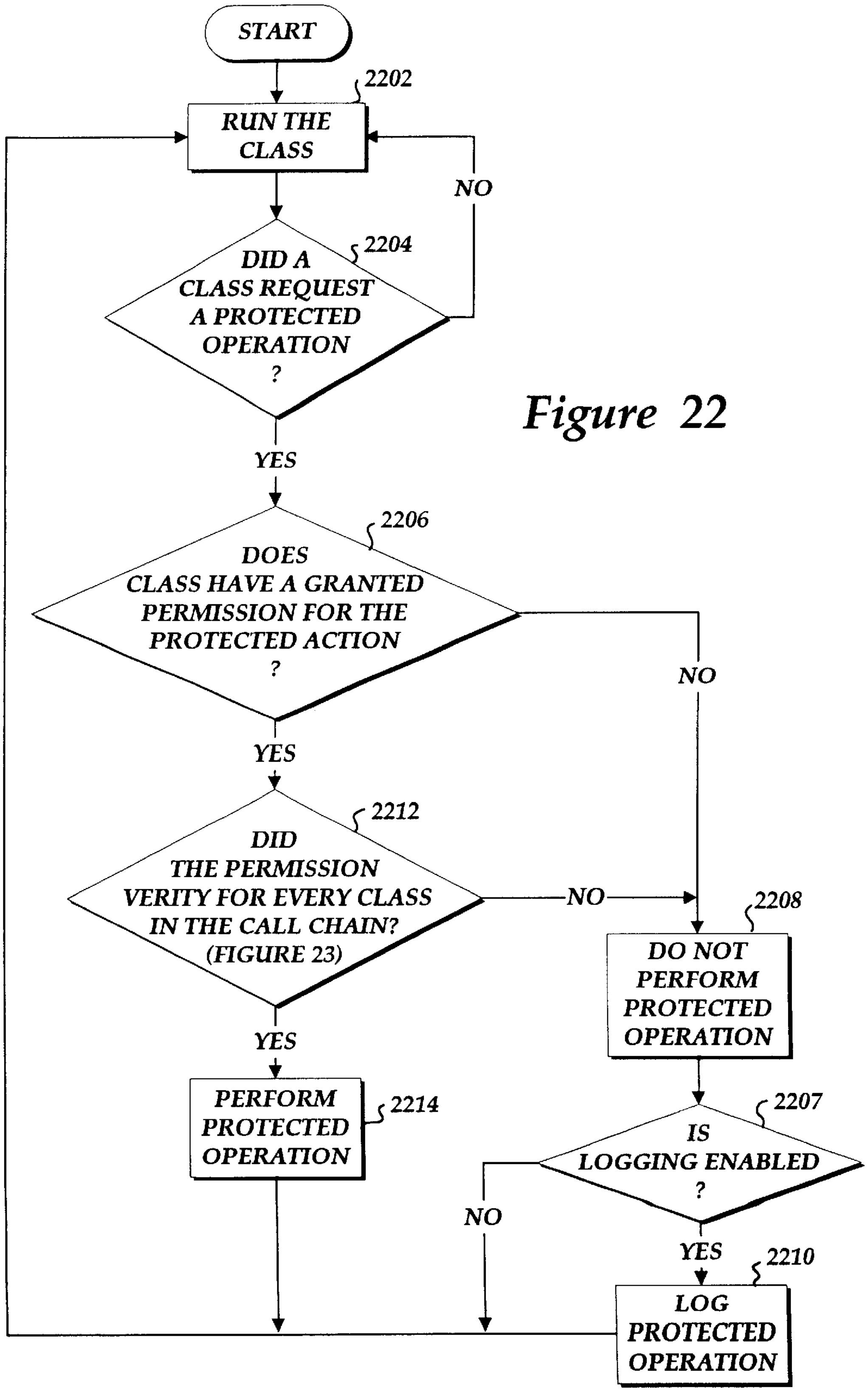

START
RUN THE CLASS
2202
DID A CLASS REQUEST A PROTECTED OPERATION ?
2204
NO
YES
DOES CLASS HAVE A GRANTED PERMISSION FOR THE PROTECTED ACTION ?
2206
NO
YES
DID THE PERMISSION VERITY FOR EVERY CLASS IN THE CALL CHAIN? (FIGURE 23)
2212
NO
YES
DO NOT PERFORM PROTECTED OPERATION
2208
PERFORM PROTECTED OPERATION
2214
IS LOGGING ENABLED ?
2207
NO
YES
LOG PROTECTED OPERATION
2210

Figure 22

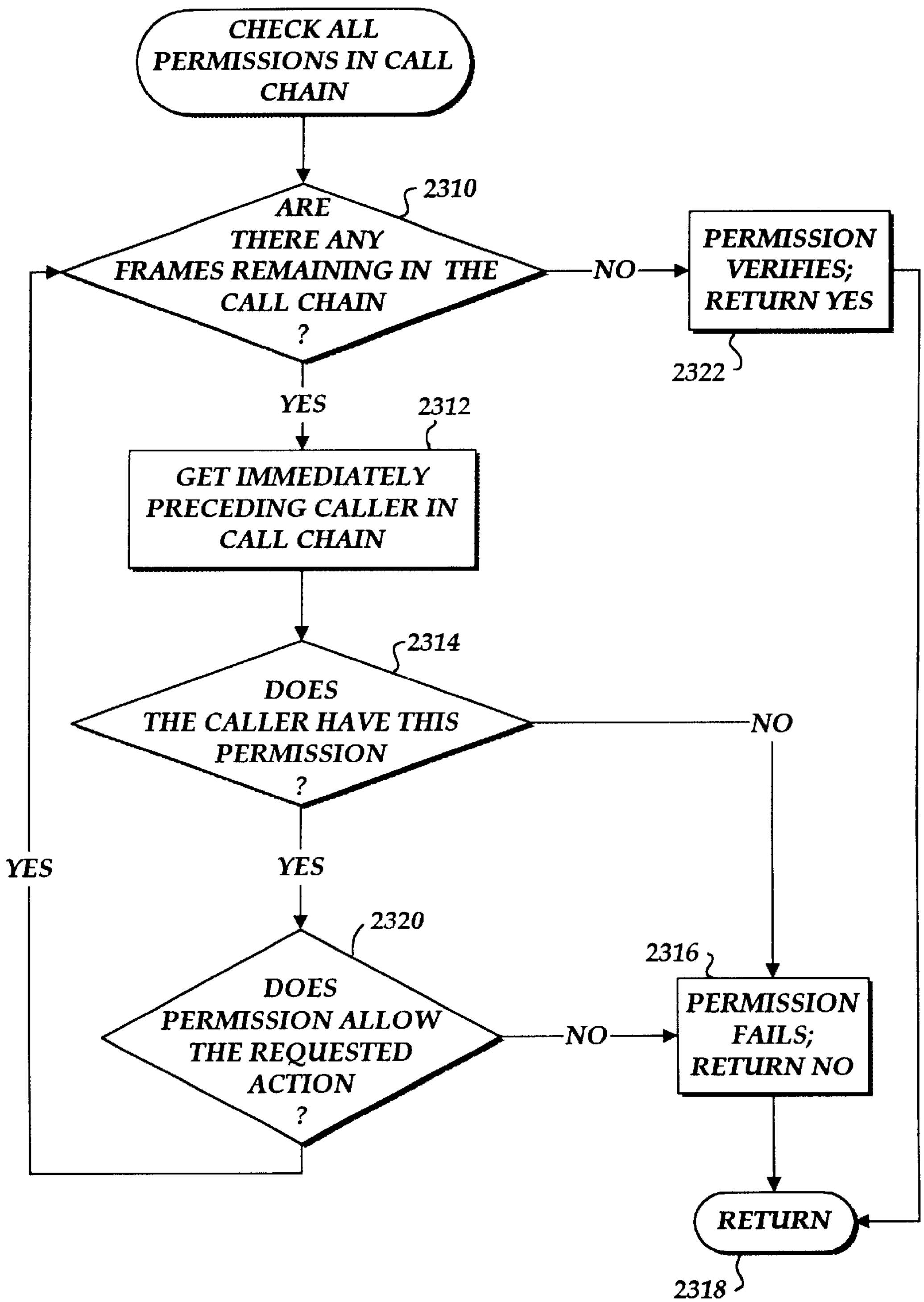

CHECK ALL PERMISSIONS IN CALL CHAIN
2310
ARE THERE ANY FRAMES REMAINING IN THE CALL CHAIN ?
NO
PERMISSION VERIFIES; RETURN YES
2322
YES
2312
GET IMMEDIATELY PRECEDING CALLER IN CALL CHAIN
2314
DOES THE CALLER HAVE THIS PERMISSION ?
NO
YES
YES
2320
DOES PERMISSION ALLOW THE REQUESTED ACTION ?
NO
2316
PERMISSION FAILS; RETURN NO
RETURN
2318

Figure 23

ADMINISTERING PERMISSIONS ASSOCIATED WITH A SECURITY ZONE IN A COMPUTER SYSTEM SECURITY MODEL

FIELD OF THE INVENTION

The present invention relates to the field of software and, in particular, to methods and systems for a comprehensive security model for managing active content downloaded from a computer network.

BACKGROUND OF THE INVENTION

In recent years, there has been a tremendous proliferation of computers connected to a global computer network known as the Internet. A "client" computer connected to the Internet can download digital information from "server" computers connected to the Internet. Client application and operating system software executing on client computers typically accept commands from a user and obtain data and services by sending requests to server applications running on server computers connected to the Internet. A number of protocols are used to exchange commands and data between computers connected to the Internet. The protocols include the File Transfer Protocol (FTP), the HyperText Transfer Protocol (HTTP), the Simple Mail Transfer Protocol (SMTP), and the "Gopher" document protocol.

The HTTP protocol is used to access data on the World Wide Web, often referred to as "the Web." The World Wide Web is an area within the Internet that stores HTML documents. The World Wide Web is made up of numerous Web sites around the world that maintain and distribute Web documents. A Web site may use one or more Web server computers that are able to store and distribute documents in one of a number of formats including the HyperText Markup Language (HTML). An HTML document can contain text, graphics, audio clips, and video clips, as well as metadata or commands providing formatting information. HTML documents also include embedded "links" that reference other data or documents located on the local computer or network server computers.

A Web browser is a client application, software component, or operating system utility that communicates with server computers via standardized protocols such as HTTP, FTP and Gopher. Web browsers receive documents from the computer network and present them to a user. Microsoft Internet Explorer, available from Microsoft Corporation, of Redmond, Wash., is an example of a popular Web browser.

An intranet is a local area network containing servers and client computers operating in a manner similar to the World Wide Web described above. Additionally, a Web browser on an intranet can retrieve files from a file system server executing on the same computer as the Web browser, or on a remote computer on the local area network. A Web browser can retrieve files on the local area network using the "FILE" protocol, which comprises file system commands. Typically, all of the computers on an intranet are contained within a company or organization. Many intranets have a "firewall" that functions as a gateway between the intranet and the Internet, and prevents outside people from breaking into the computers of an organization. A "proxy server" is one well-known portion of a firewall.

In addition to data and metadata (data about data), HTML documents can contain embedded software components containing program code that perform a wide variety of operations on the host computer to which the document is downloaded. These software components expand the inter-

active ability of an HTML document and can perform other operations, such as manipulating data and playing audio or video clips. ActiveX is a specification developed by Microsoft Corporation for creating software components that can be embedded into an HTML document. Java is a well-known programming language that can be used to develop small computer applications called "applets" and standalone software components called "classes" which are transmitted with HTML documents when they are downloaded from Web servers to client computers. JavaScript and VBScript are scripting languages that are also used to extend the capabilities of HTML. JavaScript and VBScript scripts are embedded in HTML documents. A browser executes each script as it reaches the position in the script during interpretation of the HTML document.

Some software components transferred over the World Wide Web perform operations that are not desired by a user. This may occur either because a component developer intentionally programmed the software component to maliciously perform a harmful operation, or because an unintentional "bug" in the software causes the component to perform a harmful operation. In addition to components that are transferred with an HTML document or by the HTTP protocol, files transferred to a client computer utilizing other protocols, such as FTP, may include commands that perform harmful operations.

One way in which browsers have addressed the security problem presented by potentially harmful software components is to notify the user prior to performing a potentially harmful operation while the software component is running on the host system. The user is permitted to determine, prior to each operation, whether to allow the specified operation. For example, prior to installing a Java class, a browser may display a dialog window specifying the source of the Java class and allowing the user to decide whether or not to install the specified class. Similarly, the browser may present a dialog window to the user prior to downloading a file, executing a program, or executing a script. This security procedure can result in a user repeatedly being presented with dialog windows asking for permission to perform certain operations, interrupting the user's browsing session. Faced with frequent interruptions as the software component runs, a user may respond hastily and improperly.

It is desirable to have a mechanism that allows the fine-grained administration of the permissions given to a software component, or other active content, that is downloaded from a computer network to a host system. Preferably, the mechanism would automatically administer the decision to grant or deny permissions to the downloaded active content to perform certain protected operations on the host system. The mechanism would preferably administer permissions in zones by comparing a requested set of permissions that the active content requires to run with a set of permissions that has been pre-configured in a manner that reflects the risk that active content downloaded from that zone may be harmful to the host system. Additionally, it would be advantageous if the mechanism processed the permissions required by the active content without having to run the active content and that then to stored any granted permissions with the active content so that the permission comparison need only be conducted when the active content is first downloaded. The mechanism would also preferably be able to automatically compare many different types of permissions that may defined by a wide range of expressions. Further, a preferable mechanism would provide sets of predetermined security settings that represent varying levels of trust level that can be associated with a zone, or that

provides a way for the user to configure the permission sets down to a very "fine-grained" level. The present invention is directed to providing such a mechanism.

SUMMARY OF THE INVENTION

In accordance with this invention, a system and a computer-based method of providing security when downloading foreign active content from a computer network is disclosed. Foreign active content is untrusted code that may attempt to run on a host system. The method includes configuring a system security policy to establish multiple security zones, each security zone corresponding to a set of locations on a computer network. Each zone has a corresponding security configuration that specifies the actions to be taken when a protected operation is requested by active content downloaded from that security zone. During a Web browsing session, the mechanism of the invention determines the security zone corresponding to the network location currently being browsed. Prior to performing a protected operation, the mechanism of the invention determines the action to perform, based on the current Web site's security zone, the requested operation, and the security setting corresponding to the requested operation and the Web site's zone. The Web browser displays visual information indicating the security zone corresponding to a server computer when a Web document from the server computer is being displayed.

In accordance with other aspects of this invention, during a browsing session between a client computer and a server computer, when a document is received at the client computer the browser determines if the document wishes to perform any protected operations on the client computer. If the document requires access to a protected operation, the browser determines a security setting corresponding to the zone from which the document was retrieved. Depending on the configuration of the protected operation within the security zone, the browser may perform the protected operation, prevent the performance of the protected operation, or query a user whether to perform the protected operation and selectively perform the protected operation based on the user response.

In accordance with other aspects of this invention, the client computer may be located behind a firewall, and receive active content from server computers behind the firewall and remote server computers external to, or outside of, the firewall. The browser may be configured so that one security zone does not include any server computers that are external to the firewall and so that another security zone includes only server computers that are behind the firewall. Preferably, the browser is configured so that the security zone corresponding to the server computers external to the firewall specifies a higher level of security than the security zone corresponding to server computers protected by the firewall.

In accordance with the invention, the system security policy is comprised of a number of security zones that each have an associated zone security configuration that is enforced by a security manager application on the user's computer system. Each security zone is associated with one or more server computers that are grouped into the security zone according to the likelihood that the server computers within that security zone may contain harmful active content. The user may utilize one or more predefined security zones, configure custom security zones, or do nothing and accept a default set of predefined security zones.

In accordance with other aspects of the invention, each security zone has an associated zone security policy. The user may select one of a number of predefined zone security policies, configure a custom zone security policy, or do nothing and accept a default zone security policy for the security zone. In an actual embodiment of the invention, the predefined zone security policies define levels of security that that represent "high" security (most secure), "medium" security (more secure), and a "low" security (least secure). The custom security policy permits the user to customize the zone security policy to a level defined by the user's configuration of the same security components that make up the predefined "high", "medium", and "low" pre-configured security policy options.

In accordance with further aspects of the invention, configuration of the system security policy may include the configuration of progressively "finer grain" steps or levels. The "coarsest grain" level is the configuration of one or more security zones. Each security zone has a set of configurable protected operations that can be configured. For some protected operations that regulate active content, one or more sets of permissions can be configured. Permission sets can be configured for different contexts, for instance, different permission sets can be configured for active content that is digitally signed and for active content that is not digitally signed. Each permission set can have a number of permissions and each of the permissions may have a set of parameters. At the "finest grain" of configuration, the parameters can be configured using one or more primitives.

In accordance with the present invention, at the protected operations configuration level, the user may specify whether a protected operation is allowed (enabled), is not allowed (disabled), or if the user should be prompted to determine the action that should be taken. For some protected operations, it is desirable to specify a "finer grain" configuration of the actions that are available to the protected operation when it is simply "enabled." The right to perform an action on a host system requested by a subject of a protected operation is called a permission. The configuration of the permissions available to a protected operation, at the permission configuration level, is a level "down" in the configuration of the custom zone security policy. The user may specify at the permission configuration level those permissions that define a protected operation. The permission can be granted to the protected operation (enabled), denied to the protected operation (disabled) or the user prompted for instructions when the permission is required.

In addition to configuring protected operations within security zones, the permissions that define protected operations may be configured for the context of the active content that requests the privileged operations. For instance, the user could configure the permission to be enabled when the protected operation is requested by "signed" active content, and disabled when the protected operation is requested by "unsigned" active content. For example, in an actual embodiment of the invention, the administration of permissions available to Java applets and classes is a protected operation. The user may enable or disable individual permissions for Java applets and classes in permission sets that are applied depending on the context of the active content within a zone. A permission may be configured differently in different permission sets within the same security zone. For instance, a signed applet may request access to all files on the host system. In accordance with the invention, the access all files permission may be configured in one permission set to enable the access of all files when the applet is signed and configured differently in a second permission set to disable the access to all files permission when the applet is unsigned.

In accordance with further aspects of the invention, the capabilities of each permission may be defined by a set of

"parameters" that can be configured at a parameter configuration level. In contrast to the configuration of the permissions at the permissions configuration level (a level "up") where all the capabilities of the permission are enabled, disabled, or set to require a prompt of the user, the configuration of the parameters at the parameter configuration level allows for the "fine grained" configuration of each permission. For instance, in an actual embodiment of the invention, the File I/O permission determines whether a Java applet can perform file operations on the user's computer. The File I/O permission includes parameters that determine if the File I/O permission has the right to read, write or delete files on the host computer. Parameters are defined using a number of primitive types. In accordance with the invention, a primitive is an expression that can represent values like "5", "true", "*.doc", include/exclude pairs and arrays of these types.

In accordance with the present invention, permissions for active content are grouped in one or more user permission sets that are stored in a system registry and associated with a security zone. Each security zone may have a number of differently-defined permission sets that are associated with active content having different attributes from within the same security zone. For example, in an actual embodiment of the invention, each security zone has three associated user permission sets that are stored with the zone configuration policy in the system registry: a trusted signed permission set, an untrusted signed permission set, and an unsigned permission set. If the retrieved active content is unsigned (has not been digitally signed) then the unsigned active content is granted a set of permissions corresponding to the unsigned permission set associated with the zone from which the active content was retrieved. If the retrieved active content is signed (has been digitally signed) then the present invention uses the trusted signed permission set and the untrusted signed permission set associated with the security zone from which the active content was downloaded to determine the permissions that will be granted to the active content, denied to the active content, or for which the user will be queried before the permission is granted.

In accordance with further aspects of the invention, the publisher of active content such as Java applets, classes or scripts, may externally attach a list of permissions to the active content that specifies the permissions the active content requires in order to run on the host computer. The list of permissions, or "requested permission set," is prepared by the publisher of the active content and preferably specifies the most restrictive set of permissions within which the active content can run. The present invention allows the publisher to specify each permission down to the parameter configuration level.

In accordance with another aspect of the invention, the publisher attaches the requested permission set to the outside of the active content so that the user computer does not have to run the active content in order to discover the permissions that the active content requires in order to run on the host system. The requested permission set may be included in a signed code package that also contains the computer executable instructions and other files associated with the active content. Requested permission sets may also be signed using a catalog file. A catalog file contains a manifest of hash values for other files such as cabinet files, class files, requested permissions initialization files, etc. The manifest is digitally signed, thereby authenticating the files listed in the manifest if the hash value in the manifest is equal to the newly calculated hash value of the file when it is downloaded. When the signed code package is downloaded to the user's computer, the present invention authenticates the identity of the publisher and verifies that the contents of the signed code package is identical to the information that was in the signed code package when it was signed. If the active content has not been digitally signed, the active content is granted only those permissions contained in the unsigned permission set.

If the active content has been signed, the identity of the publisher and the integrity of the downloaded signed code package are verified by the present invention. If this verification succeeds, the requested permission set is extracted from the signed code package or catalog file and then compared to the user's permission sets associated with the security zone that the signed code package was downloaded from. In an actual embodiment of the invention, the requested permission set from the signed code package is compared to the trusted signed permission set. If the requested permission set contains a subset of the permissions configured in the trusted signed permission set, the permissions requested in the requested permission set are granted and associated with the active content. If the requested permission set includes permissions, or parameters within permissions, that exceed those specified in the trusted signed permission set, the permissions in the requested permission set are compared to the untrusted signed permission set. The untrusted signed permission set may be either a deny set or a query set depending on the value of a Query/Deny flag associated with the untrusted signed permission set. If the untrusted signed permission set is a deny set and the untrusted signed permission set contains (intersects) any permissions, or parameters within permissions, that are within the requested permission set, the requested permission set is automatically denied and the active content is not run. If the untrusted signed permission set is flagged as a query set, the requested permissions must be a subset of the query set before the requested set will be granted. Any permission that is not in the query set is assumed to be in the denied set. Therefore, if the requested set is not a subset of the query set, there is at least one permission that is in the deny set and the requested set is rejected.

In accordance with further aspects of the invention, a requested permission set is automatically compared to a user permission set by the mechanism of the invention to determine if the permissions requested in the requested permission set exceed the permissions defined in the user permission set. The method and system of the invention first determines if there are any permissions in the requested permission set that are not in the user permission set.. If the permission is in the requested set and not in the permissions allowed by the user (the user permission set), the requested set is not automatically granted. If the permission is in the requested set and in the denied set then the content is not run. Next, corresponding permissions in the requested permission set and the user permission set are compared to each other. When the permissions compare themselves to each other, they compare parameter to corresponding parameter. To compare a parameter to a corresponding parameter, each primitive that defines a parameter in the requested permission set is compared to a primitive that defines a parameter in the user permission set.

Comparing the requested permission set to the user permission set involves comparing zero or more permissions in the requested permission set to zero or more corresponding permissions in the user permission set. Each permission may have one or more parameters that specify the capabilities of the permission. Each parameter may have one or more

primitives that define the parameter. The method and system of the present invention automates these progressive comparisons in a manner that produces a directional result of each comparison and maintains the direction of the result. These results are successively merged to produce a directional comparison result that can be used in later decisions to determine an action to take. For example, when comparing a requested permission set to a user permission set, it is important to be able to determine if the requested permission set is a SUBSET of the user permission set or alternatively, if the user permission set is a SUBSET of the requested permission set. In this example, it is apparent that it is important to keep track of directional nature of the comparison result because in the former case it may be appropriate to grant the permission, while in the latter case it may not be appropriate to grant the permission.

In accordance with the invention, the direction of set comparison results is maintained while the results of many comparisons that may occur on many different levels are combined to produce a cumulative directional set result. In other words, a requested permission set compares to a user permission set, which requires that requested permissions compare to user permissions, which requires that a requested permission's parameters compare with a user's permission's parameters, which requires that the primitives that define a requested permission's parameter compare to a user's permission's primitives. Each comparison results in an answer that must be combined with the answers from all other comparisons in a manner that yields a meaningful combined answer that preserves the direction of the comparison in a directional result.

In an actual embodiment of the present invention, the comparison of a primitive to a primitive produces a cumulative directional primitive result. The cumulative directional primitive result of each parameter is then combined to produce a cumulative directional parameter result. The cumulative directional parameter result of each parameter is then combined to produce a cumulative directional permission result. Finally, the cumulative directional permission result of each permission is combined to produce a cumulative directional permission set result. Because the present invention performs the comparison and accumulates the results in a manner that maintains the direction of the comparison, the cumulative directional result may be used at any level to describe the directional results of all previous comparisons to that level.

In an actual embodiment of the invention, the cumulative directional permission set result is used to determine if the permissions in a user permission set should be granted, denied, or the user should be prompted for a choice of whether to grant or deny the permissions as a set. The present invention is not limited to this implementation, however. For instance, the cumulative permission result could be used to determine if an individual permission should be granted, denied, or the user prompted for the proper action. Other decisions could be based on the cumulative directional result at "lower levels" of the accumulation.

As will be readily appreciated from the foregoing description, a system and method of providing security when downloading active content formed in accordance with the invention provides a way of selectively restricting protective operations that can be performed by active content retrieved from a computer network, such that the restrictions may vary according to the level of trust that a user has for each security zone. The invention allows the user to configure a browser to a fine grain administration of privileges allowed to active content so that the different security zones and different contexts within those security zones reflect different levels of trust for each corresponding group of network locations. Default security settings corresponding to each security zone protected operation, permission and parameter among the security zones simplifies the process of configuring the browser. Allowing a user to modify the default settings provides users with customizable security to allow for differing situations or concerns. The invention minimizes the amount of disruption that may occur during a browsing session in order to determine the user's preferences. By allowing a user to configure the security settings at a time convenient to the user, the invention increases the likelihood that the user will carefully consider the choices involved in security configurations. The ability to customize the security of the host system to a fine grain level also permits more sophisticated users, such as system administrators, to tailor the security of browsers under the administrator's control to the specific security requirements of an organization.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same becomes better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIG. **1** is a block diagram of a general purpose computer system for implementing the present invention;

FIG. **2** is a block diagram illustrating an architecture of a security method and system in a browser operating on a computer network, in accordance with the present invention;

FIG. **3** is a functional flow diagram illustrating the process of configuring the security method and system of the present invention;

FIG. **4**A is a pictorial representation of a "Internet Options" dialog window that exposes the Security tab in accordance with the present invention;

FIG. **4**B is a pictorial representation of the "Trusted sites zone" dialog window produced in accordance with the present invention;

FIG. **5**A is a pictorial representation of a "Security Settings" dialog window produced in accordance with the present invention;

FIG. **5**B is a pictorial representation of a "Security Warning" dialog window produced in accordance with the present invention;

FIG. **6** is a pictorial representation of the "Internet zone" dialog window having a configuration menu for Java permissions on a "View Permissions" tab, in accordance with the present invention;

FIGS. **7**A–E are pictorial representations of the "Internet zone" Java permissions window dialog displaying the "Edit Permissions" tab, in accordance with the present invention;

FIG. **8** is a pictorial representation of an "Edit Custom Permissions" dialog window, in accordance with the present invention;

FIGS. **9**A–G are pictorial representations of an "Edit Custom Permissions-Unsigned Permissions" dialog window, in accordance with the present invention;

FIG. **9**H is a Venn diagram illustrating an include/exclude pair primitive in accordance with the present invention;

FIG. **10** is a block diagram illustrating a signed code package having an externally attached requested permission set in accordance with the present invention;

FIG. **11** is a functional flow diagram illustrating the process of creating and distributing active content with a requested permission set externally attached in accordance with the present invention;

FIGS. **12**A–D illustrate a sample initialization (.ini) file used for the declaring of a requested permission set in accordance with the present invention;

FIGS. **13**A–C is a functional flow diagram showing the process of checking permissions requested by active content and storing granted permissions with the active content in accordance with the present invention;

FIG. **14**A illustrates the eight directional set comparison results of the present invention;

FIG. **14**B is a functional flow diagram illustrating the process of comparing permission sets to assign a directional set comparison result in accordance with the present invention;

FIG. **14**C is a functional flow diagram illustrating the process of comparing parameters within a pair of permissions to assign a directional set comparison result, in accordance with the present invention;

FIG. **15**A is a functional flow diagram illustrating the process of assigning a directional set comparison result to the comparison of inclusive Boolean primitives, in accordance with the present invention;

FIG. **15**B is functional flow diagram illustrating the process of assigning a directional set comparison result to the comparison of exclusive Boolean primitives, in accordance with the present invention;

FIGS. **16**A–B is a functional flow diagram illustrating the comparison of array primitives to assign a directional set comparison result, in accordance with the present invention;

FIG. **17** is a functional flow diagram illustrating the comparison of numerical limits primitives to assign a directional set comparison result, in accordance with the present invention;

FIGS. **18**A–Y are functional flow diagrams and associated look-up tables for the process of comparing regular expressions to assign a directional result, in accordance with the present invention;

FIGS. **18**Z–AA illustrate a plurality of example comparisons and the resulting directional set comparison result, in accordance with the present invention;

FIGS. **19**A–I are functional flow diagrams and associated lookup tables illustrating the process of comparing include/exclude pair primitives to assign a directional set comparison result, in accordance with the present invention;

FIG. **20** is a merge table used to merge two directional set comparison results to produce a single merged directional set comparison result, in accordance with the present invention;

FIG. **21** is an illustration of an example for merging directional results, in accordance with the present invention;

FIG. **22** is a functional flow diagram illustrating the process of running active content and validating permissions for protected operations in accordance with the present invention; and

FIG. **23** is a functional flow diagram illustrating the process of verifying that the permission to be used has been granted to of each class in a call chain, in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention is a system and method for configuring and enforcing a system security policy that protects a users computer from potentially harmful active content received from a server computer. In an actual embodiment, the present invention is incorporated into the Microsoft Internet Explorer (version 4.0 or later), a Web browser available from Microsoft Corporation, Redmond, Wash. The Microsoft Internet Explorer contains a help file that describes in detail many of the features of the present invention. Further details on how to access the Microsoft Internet Explorer's help file are discussed below with reference to the security configuration user interface. While the following describes the present invention in terms of an actual embodiment of the invention that is incorporated into a World Wide Web browser, the present invention is not limited to applications on the World Wide Web and may be used in any computer environment, for instance, a single computer, a local area network, an intranet, a wide area network, or the Internet.

Web browsers commonly operate within the World Wide Web, which is a portion of a global computer network known as the Internet. The Internet is comprised of a plurality of server and client computers that are interconnected for the communication of digital data. A Web site is a computer network location that stores digital data. A Web site may correspond to one or more server computers, or to a subset of the data stored on a server computer. A server computer may include multiple Web sites. For example, the data contained within a directory structure stored on a server computer may correspond to a Web site. A Web site may be identified by a specification of an Internet domain, an Internet protocol (IP) address, or a directory path.

Web sites store digital data in groupings known as documents. The process of locating and receiving digital documents from Web sites is referred to as "browsing." A Web document may contain text, image data, sound data, format data and a variety of other information known to those skilled in the art. Web documents may also have "links" or references to various information stored on the same or another Web site at other locations. Increasing, Web documents also contain, or provide links to, "active content" that may provide some functionality either within the Web document, separately as a mini-application ("applet"), as a function library or class, or even as a full-scale computer program. As used herein, active content is defined as any computer-executable instructions that are downloaded (retrieved) from a server computer and that can run on a user's (or host) computer. Examples of active content are Java applets, Java classes, HTML scripts, Java scripts, VB scripts and ActiveX controls.

While the functionality provided by active content may provide many benefits to the user, this functionality comes with some risks to the user's system. Any code that runs on a user's computer has the potential to "harm" the user's system. For instance, malicious active content may purposefully delete files from the user's hard disk. Active content does not have to be "malicious" to be harmful to a user's system—"buggy" code can inadvertently do as much harm to a user's computer as code that is purposefully designed to do harm. It is a purpose of the present invention to provide a mechanism that allows the user to draw a balance between the advantages of allowing active content to run and the risks of letting that active content run on the user's computer. In accordance with the present invention, this balance between what the active content will be permitted to do on the user's computer and what the active content will be restricted from doing can be configured down to a very "fine grain" level and associated with the zone of where the active content was retrieved from and the context in which it was retrieved. The

mechanism of the invention also enforces the security configuration once made.

The system security policy of the present invention is configured in progressively more "fine-grained levels" of configuration. As the configuration moves "down" the levels from the configuration of security zones to configuring primitives that define the parameters of a permission, the method and system of the present invention permit progressively "finer grain" control of just what the active content will be permitted to do on the user's system.

Once the configuration is completed, the invention provides a mechanism for comparing the information in the system security policy to the requirements of the downloaded active content. The invention advantageously provides for a requested permission set to be externally attached to the active content. The requested permission set specifies those permissions that the publisher of the active content asserts are necessary for the active content to run. The requested permission set is then compared by the mechanism of the present invention to one or more user permission sets (configured by the user) to determine if the requested permission set will be granted by the security manager. The method and system of the present invention to make this comparison between permission sets is described in detail below starting with the discussion of FIG. **13**A.

Exemplary Computer System and Network

As well known to those familiar with the World Wide Web, a Web browser executes on a computer, such as a general purpose personal computer. FIG. **1** and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including handheld devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network Process, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. **1**, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional personal computer **120**, including a processing unit **121**, a system memory **122**, and a system bus **123** that couples various system components including the system memory to the processing unit **121**. The system bus **123** may be any one of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) **124** and random access memory (RAM) **125**. A basic input/output system **126** (BIOS), containing the basic routines that helps to transfer information between elements within the personal computer **120**, such as during start-up, is stored in ROM **124**. The personal computer **120** further includes a hard disk drive **127** for reading from and writing to a hard disk, not shown, a magnetic disk drive **128** for reading from or writing to a removable magnetic disk **129**, and an optical disk drive **130** for reading from or writing to a removable optical disk **131** such as a CD ROM or other optical media. The hard disk drive **127**, magnetic disk drive **128**, and optical disk drive **130** are connected to the system bus **123** by a hard disk drive interface **132**, a magnetic disk drive interface **133**, and an optical drive interface **134**, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer **120**. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk **129** and a removable optical disk **131**, it should be appreciated by those skilled in the art that other types of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital versatile disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROM), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk drive **127**, magnetic disk drive **128**, optical disk drive **130**, ROM **124** or RAM **25**, including an operating system **135**, one or more application programs **136**, other program modules, and program data **138**. A user may enter commands and information into the personal computer **120** through input devices such as a keyboard **140** and pointing device **142**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **121** through a serial port interface **146** that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor **147** or other type of display device is also connected to the system bus **123** via an interface, such as a video interface **148**. One or more speakers **157** are also connected to the system bus **123** via an interface, such as an audio interface **156**. In addition to the monitor and speakers, personal computers typically include other peripheral output devices (not shown), such as printers.

The personal computer **120** may operate in a networked environment using logical connections to one or more remote computers, such as remote computers **149** and **160**. Each remote computer **149** or **160** may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer **120**. The logical connections depicted in FIG. **1** include a local area network (LAN) **151** and a wide area network (WAN) **152**. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet. As depicted in FIG. **1**, the remote computer **160** communicates with the personal computer **120** via the local area network **151**. The remote computer **149** communicates with the personal computer **120** via the wide area network **152**.

When used in a LAN networking environment, the personal computer **120** is connected to the local network **151** through a network interface or adapter **153**. When used in a WAN networking environment, the personal computer **120** typically includes a modem **154** or other means for establishing communications over the wide area network **152**, such as the Internet. The modem **154**, which may be internal or external, is connected to the system bus **123** via the serial port interface **146**. In a networked environment, program modules depicted relative to the personal computer **120**, or portions thereof, may be stored in the remote memory

storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Architecture of the Security Model

FIG. **2** illustrates the architecture of a security method and system formed in accordance with the present invention incorporated into a Web browser **204** communicating over a local area network **151** and a wide area network **152**, such as the Internet. The local area network **151** functions as an intranet, connecting client computers executing Web browsers **204** to one or more local Web server computers **208**. The local area network **151** communicates with the wide area network **152** through a firewall **212**. The firewall **212** may comprise a computer that physically connects to the LAN **151** and the wide area network **152**. Alternatively, the firewall **212** may comprise one or more computer programs executing on a computer connected to the LAN **151** and not intermediate to LAN **151** and the wide area network **152**. The firewall **212** may include a component known as a proxy server. A proxy server ensures that the topology and addressing of the local area network **151** within the firewall **212** remains hidden from any program operating on the wide area network **152**. A common function of a firewall **212** is to examine packets coming from the wide area network **152** and then either to let them through or block them according to a set of rules defined by the administrator of the local area network **151**. A primary purpose of the security measures is to exclude potentially harmful content from reaching the local area network **151** that could adversely affect the programs and other data located on the local Web server **208**.

The security provided by a firewall is essentially a "rough sort" that exclude content that may adversely affect the local web servers **208** connected to the local area network **151** without taking into consideration that it is sometimes desirable to download potentially harmful code from the wide area network. The present invention provides a method and system for allowing some of this potentially harmful content to be downloaded to the local area network while preserving the security of the local area network **151**. In an actual embodiment of the invention, the security measures of the present invention are implemented in a browser **204** that is responsible for the download. The level of access given to active content downloaded from the wide area network is configurable in progressively more "fine-grained" levels that will be discussed in detail below.

The browser **204** includes a security configuration user interface **226** that is pictorially shown in FIGS. **4–9** (discussed below). The security configuration user interface **226** allows for the configuration of user permission sets that comprise progressively more "fine-grained" definitions of the permissions that an Internet security manager **222**, which also forms part of the browser **204** will grant to active content coming from anywhere other than the local computer on which the Web browser is running. The security configuration user interface **226** allows the configuration of these actions by zone and by permission sets within zones. The configuration **227** of the security by configuration user interface **226** is stored in a system registry **224**.

A remote Web server **214** communicates over the wide area network **152** to the Web browser **204**. The remote Web server **214** may comprise one or more computer programs executing on the remote computer **149** illustrated in FIG. **1**. As should be understood by those skilled in the art of computer systems, and others, the architecture illustrated in FIG. **2** is exemplary, and alternative architectures may be used without departing from the spirit of the invention. For example, the firewall **212** is not required by the invention. Similarly, the invention does not require both the local area network **151** and the local Web server **208**. As illustrated in FIG. **1**, the client computer executing the Web browser **204** may communicate with the wide area network via a modem **154**. Additionally, a Web server may comprise a server program that executes on the same client computer executing the Web browser **204**. In such a configuration, communication between a client computer and a server computer refers to communication between programs or software components executing on the same computer.

As depicted in FIG. **2**, the Web browser **204** includes three components that perform operations in response to receiving documents from a local Web server **208** or a remote Web server **214**: an MSHTML component **216**, an SHDOCVW component **218**, and a JAVAVM component **220**. The MSHTML component **216** performs operations that control the display of an HTML page. The MSHTML component, in cooperation with additional components (not shown), also controls scripting. The SHDOCVW component **218** performs operations related to the user interface. The JAVAVM component **220** performs operations related to Java applets. The MSHTML component **216**, the SHDOCVW component **218**, and the JAVAVM component **220** perform similarly with respect to the mechanism of the present invention. Each of these components communicates with an Internet security manager **222**.

The Internet security manager **222** performs operations to determine the security zone corresponding to a Web server and to determine the permissible operations corresponding to a security zone. The Internet security manager passes security information to the MSHTML component **216**, the SHDOCVW component **218**, and the JAVAVM component **220**, when requested. The Internet security manager **222** illustrated in FIG. **2** communicates with the system registry **224**. The system registry **224** operates as a database of information pertaining to application programs that execute on the personal computer **120** (FIG. **1**). Windows 95 and Widows 98, available from Microsoft Corporation, of Redmond, Wash., are examples of operating systems that provide a system registry that is employed by application programs to store configuration information for subsequent retrieval.

The mechanism of the invention configures the Web browser to specify a plurality of zones. Each zone includes one or more Web sites, each Web site being situated on a corresponding computer network. The configuration includes information specifying a set of security settings corresponding to each zone. A security setting is a specification indicating an action to perform when a Web page from one of the zones requests a protected operation to be performed. During a Web browsing session, the mechanism of the invention determines the zone corresponding to the Web site currently being browsed. Prior to performing the protected operation, the mechanism of the invention determines the action to perform, based on the current Web site's zone, the requested operation, and the security setting corresponding to the requested operation and the Web site's zone. Depending upon the security setting, the Web browser may perform the requested operation, prevent the requested operation from being performed, or prompt the user for a decision as to whether to perform the requested operation. During the browsing of a Web site, the browser visually indicates the zone corresponding to the Web site.

As noted above, the security configuration user interface component **226** located within the browser **204** stores information pertaining to security in the system registry **224**. At

the broadest level, the security configuration user interface component **226** stores information representing the security settings corresponding to each security zone and the distribution of Web sites among the security zones. An exemplary zone configuration, denoted Zone A, is shown in block form within the system registry **224** illustrated in FIG. **2**. The zone configuration may include a plurality of zones defined by the user, by a system administrator, or shipped as a default with the product incorporating the present invention. The configuration of Zone A (FIG. **2**) includes settings for protected operations **228** that represent certain fundamental operations that if made available to active content have the potential to enable harm to the user's computer. A listing of the some of the protected operations that may be configured by the security configuration user interface **226** appears below.

For some protected operations, such as the permissions granted to active content downloaded from sources outside the user's computer, it is desirable to limit the permissions given to the active content to only those that the active content may legitimately require and that the user is comfortable granting. An actual implementation of the invention defines Java applets and classes **230** as a protected operation. The Java applets and classes **230** are assigned permissions that define the operations that the Java applet and classes **230** are permitted to access. These permissions are determined by the Internet security manager **222** by comparing them against a trusted signed permission set **232**, an untrusted signed permission set **234**, and an unsigned default permission set **236**. The untrusted signed permission set **234** has an associated query/deny flag **235** stored in the system registry **224** that indicates whether the untrusted signed permission set **234** is a query set or a deny set. As will be discussed below, the three set **232**, **234**, **236** configuration is used by the present invention to determine the permissions granted to the Java applets and classes downloaded from Zone A **226**. The security zone **226** is discussed in detail below.

The security configuration user interface **226** does not need to be part of the browser **204** and can be its own application or utility found in another application. For example, addition to the security configuration user interface **226** found in the Microsoft Internet Explorer, an actual embodiment of an alternate security configuration user interface that can also be used to edit the system security policy (FIG. **3**) is found in the Internet Explorer Administration Kit (IEAK) available from Microsoft Corporation, Redmond, Wash. The IEAK has a help file containing information on using the security user interface **226** and the configuration of permissions within zones. As will be understood by those skilled in the art of computer programming and others, alternative mechanisms for storing and accessing the security configuration information may be used. For example, the security configuration information described as residing in the system registry **224** may alternatively reside in one or more data structures internal to the application or in files.

I. Configuration of the System Security Policy

An overview of the configuration of the system security policy is illustrated in FIG. **3**. As mentioned above, the configuration of the system security policy allows the configuration of progressively more "fine-grain" configuration levels. Each configuration level is a refinement of the previous configuration level. The configuration levels are discussed in detail below, but in overview are:

A. Configure a security zone (block **310**) or accept a predefined set of security zones;
   1. Configure one or more protected operations (block **312**) associated with each security zone defined in the previous level or accept a predefined set of protected operations;
      a) Configure one or more permission sets (block **314**) for a protected operation defined in the previous level or accept a predefined set of permission sets;
         1) Configure one or more permissions for each permission set defined in the previous level (block **316**) or accept a predefined set of permissions;
            (a) Configure one or more parameters (block **318**) for each permission defined in the previous level using one or more primitives.

In the following discussion, a user is defined as anyone having the right to configure the system security policy. This can include the end user of the browser or a system administrator. As the user "drills down" through the configuration of the progressively more fine-grained definitions of the security policy, there is a corresponding level of sophistication that is required of the user. To provide for the varying levels of user sophistication, as indicated in the overview above, at most levels the user can select predefined settings that define the configuration from that level down.

A. Configuration of Security Zones

The highest level of configuration is the security zone configuration **310** exemplified by the security configuration user interface **226** dialog windows shown in FIGS. **4**A and **4**B. FIG. **4**A illustrates an "Internet Options" dialog window **402** that is presented by the security configuration user interface component **226** to configure security zones. As depicted in FIG. **4**A, a "zone" pull-down control **404** lists the different security zones. In one actual embodiment of the invention, four security zones are provided: a "local intranet" zone, a "trusted sites" zone, a "restricted sites" zone, and an "Internet" zone. The local intranet zone includes Web sites that reside on the local area network **151** (FIG. **2**) and reside on the same side of the firewall **212** as the Web browser **204**. The trusted sites zone includes Web sites that a user trusts. These are sites that a user believes have little risk that they contain files or documents that include harmful active content. Trusted sites may reside on the local area network **151** or the wide area network **152**. The restricted sites zone includes sites that a user does not trust. In general, a user does not want to allow any operations to be performed in response to files or documents received from a restricted site that may allow potentially harmful active content to be executed. The Internet zone includes by default all Web sites that are not in the local intranet zone or have not been assigned to any other zone. While this actual embodiment of the invention provides four default security zones, additional custom zones may be configured by the user. Alternative embodiments could specify more zones, or less zones, or allow a user to create or delete security zones.

The Internet Options dialog window **402** includes a zone description static text control **406** that provides a short description of the zone selected in the zone pull down control **404**. Some of the security zones are configurable, and allow a user to specify the Web sites that are included within the zone. In the actual embodiment referenced above, the local intranet zone, the trusted sites zone, and the restricted sites zone are configurable in this manner. When one of these configurable zones is selected in the zone pull down control **404** an "add sites" push-button control **418** is enabled. When a user selects the add sites push-button control **418**, the Web browser **204** presents a "Web sites" dialog window **420** that allows a user to specify the Web sites corresponding to a security zone, illustrated in FIG. **4**B and described below. The Web sites dialog window **420** provides a user with the ability to specify the Web sites corresponding to a security zone.

The title **422** of the Web sites dialog window **420** indicates the currently selected security zone from the Internet Options dialog window **402** (FIG. **4**A). To add a Web site to the currently selected zone, a user enters the Web site address and a corresponding protocol in an "add" text box **424** and then selects an "add" button **426**.

As discussed above, the Internet security manager **222** determines the security zone ID based on the address (URL) of the current Web page. The Internet security manager **222** parses the Web page address to determine the servers that are to be included in the zones according to the listing of domains within each zone. The domain has a number of sub-domains. The "top level" domain indicates a general classification or geographical location. The "second level domain" is registered to a particular user or organization. The last sub-domain is a server computer at the second level domain. For example, if the Web page address (URL) is:

http://www.microsoft.com/ie/plus/default.htm

the corresponding top level domain is:

.com

the corresponding second level domain is:

microsoft.com (registered to Microsoft Corporation, Redmond, Wash.)

and a server named "www" at microsoft.com is fully described as:

www.microsoft.com

The protocol specified in this URL is HTTP, which is used to retrieve a Web document located on the server www.microsoft.com at the path/ie/plus/default.htm. Documents can also be retrieved using other protocols such as the FTP or "FILE" protocol. For example, the corresponding address is in a local file system;

c:\documents\doc1.htm

the corresponding domain is "c:\", the document is located at path \documents\doc1.htm, and the corresponding protocol is FILE, indicating a file system protocol.

Wildcard characters may be used to specify multiple domain names. In the present invention, wildcard characters include the "*" character (indicating zero or more characters) and the "?" character (indicating any single character). For instance the regular expression "*.microsoft.com" specifies all servers at the "microsoft.com" second level domain. If the expression is "web?.microsoft.com", this indicates all servers at microsoft.com beginning with the characters "web" followed by a single character (e.g., web**1**, web**2**, webX etc.). Preferably, when the Internet security manager **222** analyzes the expression, explicit specifications take precedence over general specifications. For example, if a system is configured with "office.microsoft.com" in a first zone, and "*.microsoft.com" in a second zone, a match with "office.microsoft.com" overrides the second specification of the more general *.microsoft.com and the Web site will be considered by the Internet security manager **222** to be part of the first zone.

A user may configure the Web browser **204** so that two different protocols corresponding to the same domain reside in two different security zones. For example, referring to the addresses illustrated above, the combination of HTTP and www.microsoft.com may be configured in the trusted sites security zone, while the combination of FTP and www.microsoft.com may be configured within the Internet security zone. A user may also specify a Web site using numeric IP addresses or a numeric range to include all Web sites having an IP address within the range.

The "Web sites" list box **428** (FIG. **4**B) displays a list of Web sites that are currently configured within the currently selected security zone. To remove a Web site from a security zone, a user selects a Web site within the Web site list box **428** and selects the "remove" button **430**. By selecting (checking) check box **432**, the user is required to use the HTTPS protocol for all web sites entered. The HTTPS protocol is Web server software for Microsoft Windows NT available from Microsoft Corporation, Redmond, Wash. Among other advantages, the HTTPS protocol offers secure network connections and verification that the server purporting to send the information is actually the server sending the information.

As shown in FIG. **4**A, most of the dialog windows presented by the security configuration user interface **226** have an "OK" or "Save" button **433**, a "Cancel" button **434**, and sometimes an "Apply" button **436**. Pressing (by selecting with an input device such as a mouse or keyboard) the "OK" or "Save" button **433** causes the configuration indicated on the current dialog to be saved and exits the dialog. Pressing the "Cancel" button closes the current dialog without recording any configuration change made in the dialog. The "Apply" button **436**, when available, saves and applies the configuration but does not exit the dialog. Other dialog windows (e.g., FIG. **5**B) present a "Yes" button **514** that when pressed accepts the action suggested in the dialog window, a "No" button **516** that does not accept the action suggested in the dialog window, and a "More Info" or "Help" button **518** that accesses a help file that displays a dialog with an explanation of the dialog from which it is called. The help file may also be accessed for many of the individual controls within the dialog by selecting the control and pushing the "F1" key on the computer keyboard.

The Internet Options dialog window **402** also includes a mechanism for selecting a security level corresponding to each security zone. As depicted in FIG. **4**A, a choice of four security levels is provided for each security zone (the Internet zone is currently displayed in the dialog window **402**): high level **408**, medium level **410**, low level **412**, and custom level **414**. Each security level has a corresponding radio button control. The high security level **408** provides the most security, and excludes the greatest number of potentially damaging operations. The low security level **412** provides the lowest level of security and allows the most operations to be performed without warning the user. The custom security level **414** allows a user to customize the configuration for a security zone by specifying an action to be taken corresponding to each protected operation. The use of the custom security level is described in detail below. Alternate embodiments of the invention may include additional security levels or fewer security levels than the four levels depicted in FIG. **4**A.

For each of the security zones, a user can specify the corresponding security level. Each security zone has a default security level, which is used if not changed by a user. The default security level for the local intranet zone is medium. The default security level for the trusted sites zone is low. The default security level for the restricted sites zone is high, and the default security level for the Internet zone is medium. When a user selects a security zone in the zone pull-down control **404**, the security configuration UI component **226** indicates the corresponding security level by selecting the corresponding security level radio button **408**, **410**, **412**, or **414**. The zone security level can be reset to the default value for the zone by pressing reset button **419**.

1). Configuration of Protected Operations

In the next level down of configuration, a set of protected operations is configured for each security zone (see, FIG. **3**; block **312**). When the custom security level radio button **414**

is selected, a "settings" push-button **416** (FIG. **4**A) is enabled. Pressing the settings push-button **416** causes the security configuration user interface **226** to display a "Security Settings" dialog window **502**, illustrated in FIG. **5**. The security settings dialog window **502** includes a protected operation settings window **504**, which provides a list of protected operations that can be configured by the mechanism of the invention. For each protected operation a set of two or more corresponding settings is displayed with associated mutually exclusive radio buttons. A user can select a setting corresponding to each operation listed in the security settings dialog window **502** by selecting the associated radio button.

In one actual embodiment of the invention, the security configuration user interface, **226** provides settings for each of the protected operations listed below. Under each protected operation, the choices for each setting are listed with an "O" character representing the associated radio button for the selection.

- Script ActiveX Controls Marked "Safe for Scripting."
  - O Enable
  - O Prompt
  - O Disable
- Run ActiveX Controls and Plug-Ins
  - O Enable
  - O Prompt
  - O Disable
- Download Signed ActiveX Controls
  - O Enable
  - O Prompt
  - O Disable
- Download Unsigned ActiveX Controls
  - O Enable
  - O Prompt
  - O Disable
- Initialize and Script ActiveX Controls Not Marked As "Safe."
  - O Enable
  - O Prompt
  - O Disable
- Java Permissions
  - O Custom
  - O Low safety
  - O Medium safety
  - O High safety
  - O Disable Java
- Active Scripting
  - O Enable
  - O Prompt
  - O Disable
- Scripting of Java Applets
  - O Enable
  - O Prompt
  - O Disable
- File Download
  - O Enable
  - O Disable
- Font Download
  - O Enable
  - O Prompt
  - O Disable
- Log-On
  - O Automatic log-on only in Internet zone
  - O Anonymous log-on
  - O Prompt for user name and password
  - O Automatic log-on with current user name and password
- Submit Nonencrypted Form Data
  - O Enable
  - O Prompt
  - O Disable
- Launching Applications and Files in an IFRAME
  - O Enable
  - O Prompt
  - O Disable
- Installation of Desktop Items

-continued

  - O Enable
  - O Prompt
  - O Disable
- Drag and Drop or Copy and Paste Files
  - O Enable
  - O Prompt
  - O Disable
- Software Channel Permissions
  - O Low safety
  - O Medium safety
  - O High safety

The set of protected operations can be extended within the present invention. A setting of "enable" corresponding to an operation indicates that the operation is to be performed, when requested, without warning the user. A setting of "disable" indicates that the corresponding operation is not to be performed. A setting of "prompt" indicates that, when the corresponding operation is requested, the Web browser should query the user for instructions or whether to proceed with the operation.

FIG. **5**B illustrates an exemplary "security warning" dialog window **510** that is displayed in response to a request to perform an operation having a corresponding "prompt" setting. As illustrated in FIG. **5**B, the security warning dialog window **510** preferably informs the user of the operation to be performed and the current Web site that is requesting the operation. The user can answer yes or no to indicate whether the operation is to be performed. As depicted in FIG. **5**B, in one actual embodiment, the security warning dialog window **510** includes an "always trust software" checkbox **512**. When a user selects this checkbox, all software that is properly digitally signed from the specified source is considered to be "trusted software."

The security settings dialog window **502** (FIG. **5**A) also includes a "reset" push-button **506** and a "reset to" pull-down control **508**. When a user presses the reset button **506**, all of the settings corresponding to the protected operations in the custom security level are reset to the security level specified in the "reset to" pull-down control **508**. The user can then make changes to individual settings in the protected operation settings control window **504**.

Administering Permissions in Zones

a) Configuration of Permission Sets for Certain Protected Operations

For certain protected operations, it is advantageous to provide for a more "fine grained" configuration of security policy than the "enable", "disable" and "prompt" configuration options discussed above. The administration of active content from zones that are not fully trusted is an example of when fine grained configuration is particularly beneficial. The purpose of active content has progressed from displaying animation in Web documents to providing useful features and utilities that the user may wish to use. In general, however, the more functionality offered by the active content the more access and control that the active content must have to the host system.

Giving access and control to active content implies risk to the host system that the active content will perform some harmful action. The present invention allows the user to balance the risk of the active content performing harmful action versus the reward of the active content as advertised and to configure a security policy accordingly. By associating a security policy with a zone from which the active content is downloaded, the user can effectively assign a certain security policy to a group of Web sites having active content that pose similar risk.

Returning to FIG. **3**, the fine-grained administration of the security settings for the individual protected operations is illustrated in a protected operations configuration block **312**. Protected operations are defined by permissions grouped in permission sets. Permissions are configured in permission sets for use in the administration of permissions within zones, which is discussed in detail below (see FIG. **3**, block **314**).

The individual permissions for a protected operation associated with a security zone are configured in a block **314**. In FIG. **5**A, the configuration dialog window **504** shows a protected operation for Java classes and applets **514**. The Java protected operation **514** has a set of constituent permissions which determine the capabilities that will be allowed to downloaded Java active content from the security zone being configured. At the configuration of the permissions level **314** (FIG. **3**), the user can specify the low safety default set of permissions **518**, the medium safety default set of permissions **520**, or the high safety default set of permissions **522**. The user can also select to disable any Java content **524** or to create a custom set of permissions **526**. The selection of the custom set of permissions **526**, low safety set of permissions **518**, medium safety set of permissions **520**, high safety set of permissions **522** or to disable Java active content **524** altogether is accomplished by selecting the radio button associated with each of these entries.

In FIG. **5**A, the radio button associated with a custom set of Java permissions is shown as selected. The selection of the custom set of permissions **526** radio button exposes a Java custom settings button **530**. The Java custom settings button **530** is pressed in order to reach an Internet zone configuration screen **610** illustrated in FIG. **6**. The Internet zone configuration screen **610** includes a view permissions tab **612**. The view permissions tab **612** exposes a hierarchical listing of the permissions associated with the Java applets and classes protected operation in three permission sets. The first permission set is displayed as the permissions given to unsigned content permission set **616**. The second permissions set is the permissions that signed content are allowed permission set **618** and the third permissions set are the permissions that the signed content are denied **620**. Under each of these three permission sets **616**, **618**, and **620** are a list of the configurable permissions in the permission set. Each permission has a set of parameters that define the scope of the permission. The hierarchical display in the permission listing window can be expanded and collapsed to reveal more or less information as desired by the user using a treeview control known to those skilled in the art. For instance, it is possible to expose below the file I/O permission the read-from file URL code base parameter **624**. The read-from file URL code base parameter **624**, in turn, can be opened to expose the setting of the parameter which is indicated to be "OK" **626**.

The permissions that may be configured for unsigned content **616** are:

File I/O
Network I/O
User Interface Access
System Properties
Reflection
Threads

A similar set of permissions is listed for the permissions that signed content are allowed permission set **618** with the additional permissions:

Client Storage
User File I/O

The permissions that signed content is denied permission set **620** indicate that no permissions have currently been specified.

Custom permission sets may be defined for certain protected operations. In an actual embodiment of the invention, custom permission sets may be defined for Java applets and classes. However, the present invention also contemplates alternative embodiments for protected operations that regulate other active content. Permissions within each permission set **616**, **618**, and **620** are independently configurable.

1). Configure Permissions Associated with a Permission Set

The next level down in the progressively more fine-grain configuration of the system security policy is to configure the permissions associated with each permission set (see FIG. **3**, block **316**). FIGS. **7**A–E illustrate the user configuration interface **226** for the configuration of individual permissions within a permission set. A detailed description of the permissions that may be configured for the invention as actually implemented in the Microsoft Internet Explorer are defined in detail in several published sources available to software developers through the Microsoft Web site (www.microsoft.com) and the Microsoft Developer Network ("MSDN") subscription service available from Microsoft Corporation, Redmond, Wash. on CD-ROM. One of these published sources is entitled "Trust-Based Security For Java" <mk:@ivt:pdinet/good/java/htm/trust__based__security.htm> (MSDN Library CD, Apr. 19, 1998), incorporated herein by reference.

At this level of configuration, permissions can be configured for signed content and unsigned content permission sets. Signed content means content that has been digitally signed in such a manner that the integrity of the content and the identity of the publisher is guaranteed. The content is unsigned if the content does not have a digital signature. The creation and components of a digital signature are discussed in detail below.

The configuration of permissions indicated in FIGS. **7**A–E allows the user to disable or enable groups of permissions or individual permissions. The edit permissions user interface **702** is exposed by selecting the edit permissions tab **704**. The permissions are grouped under the permissions that will be configured for the unsigned content and a separate set of permissions that will be assigned to signed content. The permissions are displayed in a permission display window **706** which displays a hierarchy of the following permission configuration options:

- Unsigned Content
  - Run Unsigned Content
    - O Run in a sandbox
    - O Disable
    - O Enable
    - O Additional Unsigned Permissions
      - Access to all Files
        - O Disable
        - O Enable
      - Access to all Network Addresses
        - O Disable
        - O Enable
      - Execute
        - O Disable
        - O Enable
      - Dialogs
        - O Disable
        - O Enable
      - System Information
        - O Disable

-continued

- O Enable
- Printing
  - O Disable
  - O Enable
- Protected Scratch Space
  - O Disable
  - O Enable
- User Selected File Access
  - O Disable
  - O Enable

• Signed Content
  - Run Signed Content
  - O Prompt
  - O Disable
  - O Enable
    - Additional Signed Permissions
    - Access to all Files
      - O Prompt
      - O Disable
      - O Enable
    - Access to all Network Addresses
      - O Prompt
      - O Disable
      - O Enable
    - Execute
      - O Prompt
      - O Disable
      - O Enable
    - Dialogs
      - O Prompt
      - O Disable
      - O Enable
    - System Information
      - O Prompt
      - O Disable
      - O Enable
    - Printing
      - O Prompt
      - O Disable
      - O Enable
    - System Information
      - O Prompt
      - O Disable
      - O Enable
    - Printing
      - O Prompt
      - O Disable
      - O Enable
    - Protected Scratch Space
      - O Prompt
      - O Disable
      - O Enable
    - User Selected File Access
      - O Prompt
      - O Disable
      - O Enable

Standard security for Java active content has been to run the code in a "sandbox" that provides very limited access to the host system resources. The user can elect to run unsigned active content in the sandbox by selecting radio button **708**. The user can also choose to disable all permissions for unsigned content by selecting radio button **710**. When the disable unsigned content radio button **710** is selected, the ability of the user to enable or disable individual permissions for running unsigned content is also disabled by "graying out" the remaining radio buttons under the additional unsigned permissions listing **712**. Similarly, if the user chooses to enable all permissions for unsigned content by selecting the unsigned content enable radio button **714**, all permissions are enabled for the unsigned content and the radio buttons are "grayed out" so that the user is unable to specify whether individual permissions for the unsigned content are enabled or disabled.

If the user desires to run the unsigned content in the sandbox but to provide additional unsigned permissions, the user selects the run in sandbox radio button **708**. Individual permissions generally indicated by **716** can then be individually enabled or disabled by selecting the corresponding radio button. For instance, if the user desires to allow the active content downloaded from the security zone to be able to print, the user selects the enable radio button **718** under the printing permission **720** to enable printing.

As with most levels of granularity in configuration of security settings, it is possible to reset all of the actions to the default values provided in a high security, medium security, and low security permission defaults by selecting one of those options in the reset drop-down box **722** and pressing the reset button **724**. The permissions may also be reset to a saved permissions set **726** by selecting the saved permissions option **726** in the reset to drop-down box **722** and pressing the reset button **724**. A dialog inquires if the user would like to save the configuration as a saved permission set when the user exits the dialog window shown in FIGS. 7A–E. The configuration is written to the system registry **224** and the dialog window **702** is closed if the user invokes the "OK" button. If the user invokes the "Cancel" button **740**, the dialog window **702** is closed and the new configuration is not saved in the system registry **224**.

The configurations of the permissions for signed content is illustrated in FIGS. 7C–E. Signed content is inherently more trustworthy, but not necessarily trusted, because the code has been digitally signed by an identifiable publisher and the digital signature guarantees that the downloaded content is exactly what the publisher originally published. All permissions in the run signed content list **726** can be enabled by the user by selecting the enable signed content permissions radio button **728** or disabled by selecting the run signed content disable radio button **730**. As described above with regard to the unsigned content, enabling or disabling the signed content by radio button **728** and **730** enables or disables all of the permissions listed under the additional signed permissions list **732** by graying-out the radio button for the individual permissions.

If the user wishes to be prompted before signed content is allowed to run, the user selects the signed content prompt radio button **734** and then individually configures the permissions within the additional signed permissions list **732**. For instance, if the user wishes to allow signed content to print on the host system, the user will select the printing enable radio button **736**. The user can deny the signed content the right to print by selecting the disable radio button **738** or can request that the system prompt the user before running any signed content by selecting the printing prompt radio button **740**. If the prompt radio button **740** is selected, a user prompt would appear when the signed content is loaded in a user interface screen similar to the warning shown in FIG. **5**A. The user interface screen identifies the publisher of the signed software and the permission requested. The security warning user interface **510** can also warn that permission should only be granted if the user trusts the publisher identified in the security warning as verified by the digital signature.

The configuration user interfaces **226** for configuring custom permission sets is exposed by pushing the advanced edit button **736** (FIGS. **7**A–E), which causes the security configuration user interface **226** shown in FIG. **8** to display. The edit custom permissions user interface **810** permits editing of the permission parameters within three permission sets associated with each security zone: unsigned permissions set **812**, trusted signed permissions set **814**, and untrusted signed permissions set **816**. The unsigned permissions set **812** define a set of permissions that are granted to

all unsigned content from the associated security zone. By selecting an unsigned content fully trusted check box **818**, all permissions are granted to all unsigned content originating from the associated security zone. Because this option allows unsigned content full access to the host system, it is not recommended for zones such as the Internet zone where there can be significant amounts of anonymous (unsigned) malicious code. If the user selects the unsigned content fully trusted check box **818**, a warning dialog box is displayed advising the user that this option is not recommended and offer the user the option to reconsider this important choice.

Signed permissions are grouped into two permission sets associated with each security zone: trusted signed permissions **814** and untrusted signed permissions **816**. Trusted signed permissions **814** are permissions granted to signed content from that the user feels confident to allow to run without any user approval. Generally, permissions configured in the trusted signed permission set **814** should be more restrictively granted in permission sets associated with zones that are less trusted such as the Internet zone. The user may select a signed content grant all permissions check box **820** in order to allow all signed content within the associated zone to have all permissions. This unrestricted access to the host system is not recommended and the user will be presented with a warning dialog advising the user that this selection is not recommended and giving the user the chance to cancel.

The untrusted signed permissions set contains the permissions that the user wishes to be granted to signed active content that the user either does not wish to run and outright deny or alternatively wishes to be prompted before granting the permission. The user can select whether to ask for approval of all permissions configured as untrusted in the untrusted signed permission set by selecting an untrusted permissions prompt radio button **822**. Alternately, the user can automatically refuse untrusted permissions (without being prompted) by selecting an untrusted permissions deny radio button **824**. If the user selects an apply to all permissions not specifically allowed check box **826** and the untrusted permissions prompt radio button **822** is selected, the user will be prompted for all permissions that have not been specifically allowed for in the trusted signed permissions set. Alternatively, if the apply to all permissions not specifically allowed check box **826** is selected in combination with the untrusted permissions prompt radio button **824**, the user will be prompted for all permissions not specifically allowed for in the trusted signed permissions set **814**. The selection of the untrusted permissions prompt radio button **822** and the untrusted permissions deny radio button **824** sets the query/deny flag **235** in the system registry **224** indicating whether the untrusted signed permission set **234** is a query permission set or a deny permission set.

The configuration of the individual permissions within the three permission sets is accomplished by selecting a corresponding unsigned permissions set edit button **828** to edit the unsigned permissions set **812**, selecting a trusted signed permissions set edit button **830** to edit the trusted signed permissions set **814** and selecting an unsigned signed permissions set edit button **832** to edit the untrusted signed permission set **816**.

(a). Configuring the Parameters Associated with a Permission Using Primitives

The lowest level down in the fine grain configuration of the system security policy is the configuration of the parameters associated with each permission (see FIG. **3**; block **318**). Permission editing dialog windows presented by the security configuration user interface **226** for setting the individual permissions within a permission set is shown in FIGS. **9**A–G. An explanation of the function of each parameter that can be set for a given permission can be found in the Internet Explorer help file, incorporated herein by reference, which can be accessed by pushing the "More Info" button **910** in any dialog window or by selecting a permission in the dialog window and pressing the "F1" key on the keyboard. The permissions editing dialog windows include a series of permission selection tabs **912**. Selecting one of these permission selection tabs **912** displays a corresponding dialog window with a group of permissions. A files permissions dialog window **914** is shown in FIG. **9**A, a registry permission dialog window **916** is shown in FIG. **9**B, a network permission dialog window **918** is shown in FIG. **9**C, a client services permission dialog window **920** is shown in FIG. **9**D, a system permission dialog **922** is shown in FIG. **9**E, a reflection permission dialog window **924** is shown in FIG. **9**F and the custom permission dialog window **926** is shown in FIG. **9**G. The dialog windows shown in FIGS. **9**A–G are general groupings of permissions and may each provide the interface for configuring several distinct permissions.

Referring to FIG. **9**A, the file permission dialog **914** is illustrative of how a permission is configured by parameters entered into the file permission dialog **914**. The file permission has a parameter for the access type. In FIG. **9**A, the access type to be configured is shown in drop down box **915** as "read". The parameters for the read access type is defined by a primitive type referred to as an include/exclude pair. The include portion of the pair is entered into an include files text box **919** and the exclude portion of the primitive is entered into an exclude files text box **921**. The include/exclude pair entered into text boxes **919** and **921** comprise regular expression primitives. The regular expression primitives shown in the text box **919** is a string that represents the files that the active code is given permission to read while running. The exclude files text box **921** contains a string that represents the files that the active code will not be permitted to read while it is running. Include files regular expressions that have previously been entered are displayed in an include files window **923**. An add button **927** is pushed in order to add an include files regular expression primitive **928** to the include files window **923**. To remove an item displayed in the include files window **922**, the user selects one or more of the include files regular expression primitive **925** and then presses a remove button **929**. Exclude files regular expression primitives **930** such as the one shown in the exclude files text box **921** are added and removed from the exclude files list window **932** by using an add button **934** and a remove button **936** in the same way just described for add button **927** and remove button **929**.

FIG. **9**H illustrates the concept of an include/exclude pair in more detail. The include files regular expression primitive defines a set of documents that the active content will be allowed to read according to this permission. If the user wishes to exclude files contained in this include files subset, the user enters a corresponding exclude files regular expression primitive **930** that defines a subset of files that are excluded from the included files set. This concept is illustrated in the Venn diagram shown in FIG. **9**H where the area within outer ellipse **938** defines an inner region **940** representing all files included in an included files set defined by the included files regular expression primitive **928**. The files excluded from the included files set is represented by the excluded files circle **942** that defines an excluded files region **944** that contains a set of files to be excluded from the included files set. In effect the included files regular expres-

sion primitive defines the set of files that are to be included in the set that the active content can access, while the excluded files regular expression primitive takes away a portion of the files specified by the included files set. For example, the included files regular expression primitive **928** shown in FIG. **9**A (report???.doc) includes all files that begin with the letters "report", any three characters represented by the "???" wildcards and having the three letter extension ".doc". The exclude files regular expression primitive **930** removes from this set the files named "report001.doc". In another example, the included files regular expression primitive is "*.txt" and the excluded files regular expression primitive is "personal*.txt". This include files/exclude files pair grants the permission to the active content to read all files that have the extension ".txt" except for those files that begin with the letters "personal".

In Internet Explorer (an actual embodiment of the invention), it is possible to obtain a description of each component shown in the dialog window shown in FIGS. **9**A–G by selecting the component and pushing the "F1" key on the keyboard **140**. If the user wishes to begin the configuration of a dialog with no prior entries in the dialog window, the user can actuate a clear button **946**. As an alternative to configuring the permissions in the unsigned permissions set manually, the user may select the high security setting by actuating a high button **948** or may select the medium security setting by actuating a medium button **950**. The parameters for the write and delete privileges may be configured in a similar manner to the read privilege by selecting write and delete in the access type drop down box **915**. The read, write, and delete permissions listed in the registry dialog window **916** FIG. **9**B) are configured in much the same manner as just described for the file dialog window **914** and, thus, are not described further.

The network permissions dialog window **918** FIG. **9**C) illustrates include/exclude pairs like those illustrated in FIG. **9**H and described above but the include/exclude pair is comprised of an array that includes hosts **952** (server computers) and corresponding ports **954**. Included hosts are inserted in an included hosts text box **958** along with their corresponding ports which are entered into an included port text box **960** and then added to the included hosts display window **962** by pushing an add button **964**. Hosts are removed by selecting the host/port pair displayed in the window **962** and pressing a remove button **965**. An example of an entry into the include hosts text box **958** would be "www*.microsoft.com" and the entry into the ports text box **960** would be 80. A corresponding entry into an exclude hosts text box **968** would be "www1.microsoft.com" with an entry in an associated exclude ports text box **970** of 80. The entries in the exclude hosts text box **968** and the exclude hosts ports text box **970** are entered into an excluded hosts display window **972** by actuating an add button **974**. Entries in the excluded hosts display window **972** are removed by selecting an entry and pressing a remove button **976**. The result of the include hosts "www*.microsoft.com:80" and the exclude hosts "www1.microsoft.com:80" is that all servers beginning with the letters "www" at the second level domain "microsoft.com" will be permitted connect access at port **80** except for the server named "www1.microsoft.com:80". Port **80** is associated with the World Wide Web connection. Other ports may be associated with other protocols and services known to those skilled in the art.

The client services dialog window **920** shown in FIG. **9**D illustrates a parameter that is defined by a numerical limit primitive **980** that is entered into the storage limit text box **982**. Other parameters are grouped by permissions in the clients services dialog **920**. Some of these permissions contain parameters that are defined using Boolean primitives. For example, if the access to roaming files check box **984** is selected, the Boolean primitive associated with the access to roaming files check box **984** indicates true. Otherwise, the Boolean primitive indicates false. The numerical limit primitive **980** indicated in the storage limits text box **982** defines the upper limit of storage in kilobytes that the active content will be permitted to store on the host system. The system dialog box window **922** shown in FIG. **9**E in the suffixes text box **986** contains a regular expression that refers to "applet".

The user can define custom permission in the custom dialog window **926** (FIG. **9**G) by specifying a class name in the class name text box **988** and the associated parameters in the text box **990**. These values are inserted into a custom permissions window **992** by actuating an add button **994** and removed from the custom window **992** by actuating a remove button **996**. A description of the individual permissions and parameters may be found in the Internet Explorer help file and in the published references referred to above.

Returning to FIG. **3**, after the user has configured the system security policy as is described above, the configuration data is stored (block **320**) in the system registry **224**. It will be apparent to one skilled in the art that the configuration information may be stored as the changes to the configuration are made, as each level is completed by selection of the "OK" or "Apply" button, all at once after the user confirms that the changes should be accepted, or a combination of the above. The decision **322** in FIG. **3** illustrates that multiple zones may be configured under the system and method of the present invention.

Declarative Permissions

The present invention includes a method and system for administering declarative permissions. Declarative permissions refer to the ability of an active content publisher to associate a requested permission set with a Java class, Java applet or other type of active content that requires certain permissions in order to run on a host system. Declarative permissions enable a publisher of active content to request only the permissions that are necessary to run the class. This capability enhances the security of the host system by ensuring that the class will only have access to protected operations that are absolutely necessary in order to run the class. As will be discussed in detail below, the requested permission set, which is declared by the publisher of active content, is compared to the trusted signed permission set **232** and the untrusted signed permissions set **234** associated with zone **226** from which the active content is downloaded to determine which permissions will be granted automatically to the class, which permissions will automatically been denied to the class, and which permissions will require a prompt to the user for instructions before the class is allowed to run on the host system.

FIG. **10** illustrates a signed code package **1010** that contains the computer executable instructions **1020** for one or more classes, objects, scripts, executables, or other type of active content. The signed code package **1010** can also contain other types of associated files **1030** such as data files, bitmaps, and audio files. In accordance with the invention, a publisher of active content attaches a requested permission set **1040** to the signed code package **1010**, which also contains a digital signature that identifies the publisher **1045**. While an actual embodiment of the invention uses a signed code package **1010**, the requested permission set may be

stored separately from the active content. For instance, a catalog file contains a manifest of hash values for computer files. The hash value of the requested permission set **1040** (and files such as computer executable instructions **1020** and associated files **1030**) could be included in the manifest maintained by the catalog file and verified, as discussed below, by comparing this original hash value to a newly computed hash value of the requested permission set.

A requested permission set formatted as an exemplary initialization (.ini) file is illustrated in FIGS. **12**A–D and will be discussed below. In an actual embodiment of the present invention the signed code package **1010** comprises a cabinet file (.cab). A cabinet file is a file that contains a plurality of other files that have been compressed in a manner that the original files can be extracted at a later time. The signed code package is "digitally signed" by first computing a hash value for the code **1020**, associated other files **1030** and the declarative requested permission set **1040**. A hash value results from applying a "hash function" to the signed code package **1010** to produce a hash value **1050**. A hash function is a mathematical algorithm that transforms a digital document, such as the signed code package, into a smaller representation of the document. The smaller representation of the document is the hash value corresponding to the document.

A "secure hash function" is a hash function that is designed so that it is computationally unfeasible to find two different documents that "hash" to produce identical hash values. A hash value produced by a secure hash function serves as a "digital fingerprint" of the document. If two separately produced hash values are equivalent, one can be certain to a very high degree of probability that the documents used to produce the respective hash functions are exactly the same. Similarly, if two hash values are not the same, the corresponding documents are not exactly the same.

As discussed in further detail below, the mechanism of the invention computes a new hash value corresponding to an electronic document, and compares the new hash value to the hash value that has been included in the digital signature of the document, in order to determine whether the documents are equivalent, and therefore whether the electronic document has changed since it was published. In one actual embodiment of the invention, a secure hash function known as "MD5" is used to create hash values. The MD5 secure hash function is published by RSA Laboratories of Redwood City, Calif., in a document entitled RFC 1321.

The hash value **1050** of the signed code package **1010** as of the time it is published is digitally encrypted using a public/private encoding algorithm and made part of the digital signature of the signed code package **1010**. In other words, the private key of a publisher **1060** is used to encrypt the hash value **1050** of the signed code package **1010**. In an actual embodiment of the invention, the encryption of the hash value **1050** is accomplished using a digital certificate **1060** that is issued by a third party certificate authority. A certificate authority publishes policies and acts to grant code signing authority, also known as X.509 certificates, based on criteria established in various specifications that the certificate authority publishes. The certificate authority manages the enrollment, renewal and revocation of certificates. As part of the process in granting a certificate, the certificate authority verifies various evidence submitted by a publisher when the publisher requests a certificate to ensure that the publisher is actually the individual or entity that it says it is. The inclusion of declarative permissions in the signed code package **1010** is an extension of the Authenticode specifications available from Microsoft Corporation, Redmond, Wash. The method of signing a cabinet file in accordance with the invention is described in detail entitled "Signing a Cabinet File with Java Permissions Using Signcode", <mk:@ivt:pdinet/good/java/htm/signcode.htm> (MSDN Library CD, April, 1998), incorporated herein by reference.

An example development process for the developers of active content to take advantage of the declarative permissions capability of the present invention is shown in FIG. **11**. The publisher first develops the computer-executable instructions for the class (or other active content) (block **1110**) and then specifies the permissions that the class requires to run in a requested permissions set (block **1112**). Preferably, the permissions that the publisher requests are the minimum permissions actually required by the class to run on the host system.

An exemplary initialization (.ini) file **1202** containing a requested permissions set in accordance with the invention is shown in FIGS. **12**A–D. The requested permissions set .ini file contains a list of permissions with each permission referenced by a text string **1230***a,* **1230***b,* **1230***c,* . . . contained within square brackets. The text strings **1230***a,* **1230***b,* **1230***c,* etc., are followed by a list of setting parameters, **1232***a,* **1232***b,* **1232***c,* . . . unless the specific parameters are required. The permissions listed in the exemplary .ini file **1202** correspond to permissions that are defined for the host system, either in the predefined permission sets or in a custom permission set configured as illustrated in FIGS. **8** and **9**A–G. For instance, the file I/O permission **908** (FIG. **9**A) has a corresponding permission definition identified in the requested permissions set .ini file by the text string [com.ms.security.permissions.FileIOPermission] **1230***c* (FIG. **12**B). The file I/O request and permission includes a series of exemplary parameter entries—IncludeRead, ExcludeRead, IncludeWrite, ExcludeWrite, IncludeDelete, ExcludeDelete and ReadFileURLCodeBase. The IncludeRead parameter **1212** corresponds to the entries made in the read access include files window **923** with the individual entries **925** shown in the window separated by semicolons following "IncludeRead=" entry in the .ini file **1202**.

In the file I/O requested permission example shown in FIG. **12**B the IncludeRead parameter **1212** includes a regular expression primitive "foo.mdb" **1214** and a regular expression primitive with a wildcard "*.txt" **1216**. The ExcludeRead parameter **1218**, which corresponds to the read access exclude files window **932** in FIG. **9**A, includes a single regular expression primitive "crayon" **1220**. Those skilled in the art will recognize that the format of the requested permissions .ini file is not important to the invention and could be accomplished by using many other formats as long as the implementation of the Internet security manager **222** recognizes the format of the requested permissions set. A description of the remaining permissions and parameters illustrated in FIGS. **12**A and B may be found in a published source entitled "Java Permissions .INI Values Reference" <mk:@iut:pdinet/good/java/htm/sampleinitable.htm> (MSDN Library CD, April 1998), incorporated herein by reference.

Returning to FIG. **11**, once the publisher develops the requested permissions set in the appropriate format, the publisher attaches the requested permissions set **1202** to the class (block **1114**). The requested permission set **1202** is preferably externally attached to the class in the signed code package **1010**. "Externally attached" means that the requested permission set is not part of the compiled code of the class and can be independently retrieved from the class

for processing by the Internet security manager **222**. External attachment of the requested permission set provides the significant advantage that the code does not need to be run by the host system in order to determine the permissions required by the class. Any code permitted to run on a host system is a security threat. This threat is completely avoided by adding a textual declaration of permissions outside of the computer-executable instructions set. Another advantage provided by the invention is that the code does not need to be recompiled in order to charge the permissions requested in the requested permission set **1202**. In an actual embodiment of the invention, the requested permissions set is included as an authenticated attribute to the digital signature.

After the publisher has attached the requested permissions set to the class (block **1114**), the entire signed code package **1010**, including the code **1020** and the requested permission set **1202**, is digitally signed (block **1116**) using the private key associated with the publisher's certificate to encrypt the hash value calculated for the signed code package **1010**. When the digital signature is decrypted using the public key associated with the private key by certificate holder, the identity of the publisher **1045** can be determined with a high degree of certainty because of the publisher's certificate registration **1060** with the certificate authority. In addition to the certificate **1060** that identifies the publisher, the digital signature is further authenticated by a second encryption of the digital signature using a certificate owned by the certificate authority. One significant advantage of a digital certificate is that the digital certificate establishes the identity of the publisher in a manner that the publisher can not repudiate the published code because it was published using the publisher's private key and decrypted using the corresponding public key.

When the digital signature is decrypted using the public key associated with the digital certificate **1060**, the original hash value of the signed code package **1010** that was computed when the signed code package **1010** was published is obtained. The hash value of the signed code package is then recomputed by the Internet security manager **222** and compared to the original hash value included with the digital signature. If they match, the user can be assured with a high degree of certainty that the contents of the signed code package, including the computer-executable instructions **1020** of the class and the requested permissions set **1040**, have not been altered since the class was published (in block **1118**) by the publisher.

The method and system for processing the requested permission set is set forth in the functional flow diagram shown in FIG. **13**. The Internet security manager **222** first determines the zone that the class was downloaded from (block **1310**) when a class or other active content is first downloaded. The downloaded class **1010** is then checked in a signature verification decision (block **1312**) to determine if the downloaded class has been digitally signed. If the class has not been digitally signed, then the Internet security manager **222** retrieves (block **1314**) the unsigned default permissions set **236** for the zone **227** from which the class was downloaded. The Internet security manager **222** then grants the permissions contained in the unsigned permissions set (block **1316**). The permissions granted (block **1316**) are stored with the class (block **1318**; FIG. **13**C). The storage (block **1318**) can either be temporary, if the class is temporarily downloaded, or permanent if the class is to be stored on the host system.

Returning to the digital signature decision (block **1312**) in FIG. **13**A, if the Internet security manager **222** determines that the class has been digitally signed, then the identity of the publisher is determined from the signing certificate (block **1320**). A decision (block **1322**) determines if the signature verifies for the asserted publisher. If the signature of the publisher does not verify, the process fails (block **1324**; FIG. **13**C), the code is not loaded, run or stored on the host system and no permissions are granted.

If the signature does verify as the publisher's (block **1322**; FIG. **13**A), then the Internet security manager **222** computes a new hash value for the signed code package **1010**, as it has been received from the zone, and compares (block **1326**) this hash value to the original hash value **1050** (FIG. **10**) contained in the digital signature of the signed code package **1010**. A failure of the new and the original hash values to match indicates that the contents of the signed code package **1010** have been altered or corrupted since the time that the signed code package **1010** was published by the publisher (block **1118**; FIG. **11**), the process fails (block **1324**; FIG. **13**C) and the class is not loaded, stored or run and no permissions are assigned.

If the original and the new hash values do match, as determined in the hash value comparison decision (block **1327**), the set of permissions that the class requires to run is obtained. First, a decision (block **1330**) determines if a requested permission set is externally attached to the class. If a permission set is not attached to the class, the default set of permissions is processed (block **1314**), as is described above. If the decision (block **1330**) determines that a requested permission set is attached, the requested permission set is retrieved (block **1331**). Thereafter, the Internet security manager retrieves the trusted signed, i.e., granted, permission set **232** that is associated with the zone **226** from which the class was downloaded from the system registry **224** (block **1332**). The granted permission set **232** is then compared to the requested permissions set (block **1334**). If the requested permission set is a subset of the granted permission set, as each permission is defined by its parameters and primitives (block **1336**; FIG. **13**B), the permissions requested in the requested permission set are granted (block **1338**; FIG. **13**C).

If the requested permissions set **1202** is not a subset of the trusted signed, i.e., granted, permissions set **232** the Internet security manager determines in a query/deny set decision (block **1340**) whether the untrusted signed permission set **234** is a query set or a deny set. In an actual embodiment of the invention, the Internet security manager **222** determines whether the untrusted signed permission set **234** is a query set or a deny set by reading a query/deny flag **235** that is stored in the system registry **224** and associated with the untrusted signed permission set **234**. If the query/deny set decision (block **1340**) determines that the untrusted signed permission set **234** is a query set, the untrusted signed (query) set **234** that is associated with zone **227** from which the class was downloaded is retrieved from the system registry **224** (block **1342**). The Internet security manager **222** then compares the requested permissions set **1202** to the untrusted signed (query) permission set **234** (block **1344**). If the requested permissions set is not a subset of the untrusted signed (query) permission set **234**, the loading of the class fails (block **1324**; FIG. **13**C), the class is not loaded or run and no permissions are assigned.

If the requested permissions set is a subset of the untrusted signed (query) permission set **234**, a dialog window displays the publisher and requested permissions to the user for approval or disapproval (block **1348**, FIG. **13**C). (See FIG. **5**B for an exemplary prompt dialog box). A user decision not to approve the requested permissions is detected in a query approval permissions decision (block **1350**), resulting in the

process failing (block **1324**), the class not loading or running, and no permissions being granted. A user decision to approve the request for the permissions is detected in a query approval decision (block **1350**), resulting in the requested permissions being granted (block **1338**). Permissions that are granted (block **1338**) are stored with the class in memory and then stored (block **1318**) with the class on the user's local storage if the class is to be retained over time.

If the decision (block **1340**; FIG. **13**B) is that the untrusted signed permission set **234** is a deny set, the untrusted signed (deny) permission set **234** is retrieved (block **1352**; FIG. **13**B) from the system registry **224** and compared (block **1354**) to the requested permission set **1202**. If any of the permissions in the requested permission set **1202** are also in the untrusted signed (deny) permission set **234**, this is detected in a decision (block **1356**; FIG. **13***b*), the process fails (block **1324**; FIG. **13**C), and the class is not loaded, run or saved. If none of the permissions in the requested permission set **1202** are in the untrusted signed (deny) permission set **234** (block **1356**; FIG. **13**C), the user is queried (block **1348**) to approve or disapprove of the requested permission set.

While actual embodiment of the present invention uses unsigned, granted, and denied/query permission sets, the invention also contemplates alternative implementations for controlling active content based on set operations of other permission set types. For example, one or more sets can be used to implement control over active content of different types, i.e. signed, unsigned, size, or any other attribute.

In an actual embodiment, the method and system of the present invention is realized using an object-oriented programming paradigm. An object-oriented programming paradigm views blocks of computer-executable instructions and data as a collection of discrete objects that interact with other objects. One of the primary benefits of object-oriented programming is that the objects can easily and affordably be adapted to meet new needs by combining them in a modular fashion. An object is a unit of code comprising both routines and data (also called methods and properties) and is thought of as a discrete entity. The structural foundation for an object-oriented language is the object model. The goals of this model are encapsulation, persistence, polymorphism, abstraction, concurrency, and typing. The component object model (COM), the distributed component object model (DCOM), and object linking and embedding (OLE) produced by Microsoft Corporation of Redmond, Wash., are examples of object models. The present invention may be practiced under any of these object models or many others that are well known to those skilled in the art.

Objects communicate with each other through interfaces. Each object may have multiple interfaces. An interface exposes and defines access to the object's public properties and methods. For instance, in Microsoft's COM, all objects are required to support the IUnknown interface. The IUnknown interface includes a method named QueryInterface through which other objects in the global namespace (or a distributed namespace in a distributed system) can request and receive pointers to the objects' other interfaces. One of the primary advantages to interfaces is that a client object can continue to access the methods of a server object that are exposed through the interface regardless of whether the underlying code in the object is updated or changed for another reason.

In an actual embodiment of the invention, permissions are modeled as objects, or components of active code, that are attached to the downloaded classes according to the method and system described above with reference to FIG. **13**. When a downloaded class is instantiated as an object, the permission objects associated with that class intercepts all protected operation requests received from the object while it is running and allow access to the underlying system objects only if allowed by the permission objects that have been granted.

Set Comparisons

The method and system of the present invention automates the comparison of a requested permission set with a user permission set to produce a directional set comparison result that is used to decide whether to grant permissions to active content, deny permissions to active content, or to prompt the user for instructions on what to do. In an actual embodiment of the invention, a requested permission set **1202** is compared to two user permission sets—the trusted signed permission set **232** and to the untrusted signed permission set **234**. While the following discussion discusses an actual embodiment that compares permission sets, the present invention is not limited to this specific application and may be used in any application that would benefit from determining a directional set comparison result at any of a variety of different points (or levels) during the comparison.

A brief review of the preceding discussion might be helpful before continuing. The user configures a system security policy that is stored in the system registry **224** of a host computer. The system security policy is divided into configurable zones that group locations on a computer network according to the user's perception of the risk of running active content downloaded from that zone on the user's computer. Each zone **226** has a number of configurable protected operations **228**. One or more of these protected operations **228** may administer active content downloaded from the zone **226**. Each protected operation **228** may have a plurality of permission sets (e.g., **232**, **234**, **236**) that may be defined for use in different contexts, such as applying a first and a second permission set to downloaded active content that is digitally signed and applying a third permission set when the active content is not digitally signed. Permission sets contain the definition of one or more configurable permissions (e.g.; those shown in FIG. **6**). Each permission is defined by one or more configurable parameters (FIGS. **9***a–g*) and each parameter is defined by one or more primitives that can represent values like "*.txt", "mydoc.doc", "True", "**80** ", "applet" and arrays of the various types of primitives.

A requested permission set **1202** is created by a third party publisher. The requested permission set **1202** contains the permissions that the publisher requests to be granted on the host system. The permissions in the requested permission set **1202** correspond to the permissions that can be configured in the user permission sets (e.g., **232** and **234**). Like the permission configuration in a user permission set, the permissions in a requested permission set **1202** may be configured to the same "fine grained" level, i.e. down to the primitives level.

The mechanism of the invention maintains the direction of the comparison result during the comparison of permission sets because the comparison seeks to determine the result of comparing a "superior" set to an "inferior" set in the sense that the user permission set (the "superior set") limits the permissions that may be granted to the requested permission set (the "inferior set"). For instance, the trusted signed permission set **232** is "superior" to the "inferior"

requested permission set **1202** in that the permissions requested in the requested permission set **1202** must be contained in the trusted signed permission set **232** before the requested permissions will be granted. The comparison must also take into account that the parameters in a requested permission must be contained in the parameters in the corresponding user permission set, and that the primitives that define the requested parameter must be contained within primitives that define the corresponding user parameter. This same concept can be represented by set relationships, i.e., the requested permissions in the requested permission set must be a subset of the user permissions in the user permission set, the requested parameters in the requested permission must be a subset of the user parameters in the user permission and the requested primitives that define the requested parameters must be a subset of the user primitives that define the user parameters. In order to determine if the requested permission set is a subset of the user permission set, and not vice versa, a directional set comparison result must be maintained, and it is preferable that a method is provided to combine the directional set comparison results at and between the various levels of configuration to form a cumulative directional set comparison result that can be used to make decisions.

Directional set comparison results can also be determined under the present invention for comparisons that seek to determine how requested permission sets and user permission sets intersect (have common members). For instance, in an actual embodiment of the invention, if a requested permission (parameter, primitive) from the requested permission set has any part in common with a corresponding user permission (parameter, primitive) in the untrusted signed (deny) permission set, then the permission will be denied by the Internet security manager **222**.

The present invention creates a meaningful cumulative directional set comparison result that tracks the relationship of the "superior" and the "inferior" permission set as the comparisons are made up the various levels. The following discussion will explain in detail the method and system of the invention for making these comparisons, creating the directional comparison results, and merging the directional comparison results into a single meaningful cumulative set comparison result that can be used by the Internet security manager **222**, or another process, to automatically determine an action that should be taken when processing permissions.

In an actual embodiment of the invention, Java applets and classes **230** are defined as a protected operation **228**. The Java applets and classes protected operation in this actual embodiment has three permission sets per zone: the trusted signed permission set **232**, the untrusted signed permission set **234**, and the unsigned permission set **236**. Each of these permission sets **232**, **234**, and **236** is independently configurable by the user through configuration user dialogs such as those illustrated in FIGS. **9**A–G or by implicitly or explicitly accepting the default configurations that are provided. The mechanism of the invention for comparing the trusted signed permission set **232** and the untrusted signed permission set **234** to the requested permission set **1202** is described in detail below.

As an example of the actual embodiment of the invention to give context to the discussion below, the Internet Zone has a Java permissions protected operation **516** (FIG. **5***a*), which has a FileIO permission **908** (FIG. **9**A), which has a has a read parameter **909**. The read parameter **909** is defined by one or more include files regular expression primitives (e.g., "*.txt"). The read parameter **909** may also be defined by one or more exclude files regular expression primitive (e.g., "mydoc.txt"). Regular expressions primitives, such as are used in the include files parameter and the exclude files parameter may contain wildcards ("*" or "?"). The include files and exclude files parameters form an include/exclude pair that together define the subset of files to which the read parameter **909** applies.

To facilitate keeping track of the directional nature of the comparisons between the requested set **1410** and the user set **1412**, the method and system of the present invention define the following eight relationships depicted in FIG. **14**A:

Directional Set Comparison Result:	Explanation of the Directional Set Comparison Result	Illustrated in FIG. 14A as:
Empty	There is no items in either the user set or the requested set.	1416
Equal	The items in the user set are identical to the items in the requested set.	1414
Subset	The user set contains all of the items in the requested set, but there are some items in the user set that are not in the requested set.	1422
Superset	The requested set contains all of the items in the user set, but there are some items in the requested set that are not in the user set.	1424
Empty Subset	The requested set does not have any items and the user set does have some items.	1426
Empty Superset	The requested set has some items, but the user set does not have any items.	1428
Disjoint	The requested set has some items and the user set has some items but the requested set and the user set have no items in common.	1420
Overlap	The requested set has some items and the user set has some items but the requested set and the user set have only some items in common and each set contains one or more items that are not in the other set.	1418

FIG. **14**B is a functional flowchart illustrating the method and system of the present invention for comparing permission sets. FIG. **14**B illustrates the system and method of the invention for comparing permission sets when: it is called (block **1334** FIG. **13**A) to compare the requested permission set **1202** ("Rc") to the trusted signed (granted) permission set **232** ("Az"); when it is called (block **1344**; FIG. **13**B) to compare the requested permission set **1202** ("Rc") to the untrusted signed (query) permission set **232** ("Qz"); and when it is called (block **1356**; FIG. **13***b*) to compare the requested permission set **1202** ("Rc") to the untrusted signed (denied) permission set **232** ("Dz").

The comparison of permission sets may include some or all of the following steps, depending on how the permission sets have been configured: determining a directional permissions sets comparison result (which may include determining and aggregating one or more directional permission comparison results); determining a directional permission comparison result (which may include determining and aggregating one or more directional parameter comparison results); and determining a directional parameter comparison result (which may include determining and aggregating one or more directional primitive comparison results). At the end of the process in FIG. **14**B, the directional permissions sets comparison result is the accumulation of all of the "lower level" directional comparison results.

The first step in FIG. **14**B initializes a variable designated DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT to the value EMPTY (block **1430**). Thereafter, the process attempts to retrieve (block **1432**) a permission from one permission set and then tries to retrieve a corresponding permission from the other permission set. Then a test is

made to determine if a permission has been retrieved from the requested set (block **1434**). If no permission has been retrieved from the requested set, a test is made to determine if a permission has been retrieved from user defined set (block **1436**). If no permission has been retrieved from the user defined set, a variable designated DIRECTIONAL PERMISSION COMPARISON RESULT is set equal to EMPTY (block **1439**). If a permission has been retrieved from the user defined set, the variable DIRECTIONAL PERMISSION COMPARISON RESULT is set equal to EMPTY SUBSET (block **1438**).

The DIRECTIONAL PERMISSION COMPARISON RESULT is then combined (block **1440**) with the variable designated DIRECTIONAL PERMISSIONS SETS COMPARISON using the merge table illustrated in FIG. **20**. FIG. **20** illustrates a plurality of merger cells identified by the intersection of a column numbered **1** through **8** and a row identified by letters A through I. For example, the cell reference "H3" refers to the cell at the intersection of row H with column 3, which has the value OVERLAP. The table **2010** is used to merge directional comparison set results by finding the value of the present designated DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT (the Previous/Accumulated Merge Result) in column 1 and the value of the variable DIRECTIONAL PERMISSION COMPARISON RESULT (the New Merge Result) in row A. The intersection of the thusly found column and row in the table **2010** is a new designated DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT value. For example, if the present DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT value is EMPTY and the DIRECTIONAL PERMISSION COMPARISON RESULT value is EMPTY SUBSET, the new DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT is EMPTY SUBSET.

Returning to FIG. **14**B, if a permission has been retrieved from the requested set (block **1434**), a test is made to determine if a permission has been retrieved from the user defined set (block **1442**). If no permission has been retrieved from the user defined set (block **1442**), DIRECTIONAL PERMISSION COMPARISON RESULT is set equal to EMPTY SUPERSET (block **1444**) and combined with DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT in accordance with the table illustrated in FIG. **20** (block **1440**).

If a permission has been retrieved from the user defined set (block **1442**) the retrieved requested set permission and the retrieved user defined set permission are compared (block **1446**). How the comparison is accomplished is illustrated in FIG. **14**C and described below. The end result of the comparison is the value of the DIRECTIONAL PERMISSION COMPARISON RESULT variable, which is combined with the DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT variable (block **1440**).

If there is another permission to process (decision block **1442**), then the worker thread attempts (block **1432**) to retrieve another pair of like permissions from the permission sets that are being compared, and the sequence of steps illustrated in FIG. **14**B and described above are repeated. After all of the permissions have been processed, the final DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT is returned (block **1448**) to the block that called the FIG. **14**B process (block **1336**, **1346** or **1356**).

FIG. **21** illustrates a sequence of DIRECTIONAL PERMISSION COMPARISON RESULT, DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT combinations based on the table illustrated in FIG. **20**.

In one actual embodiment, the process in FIG. **14**B iterates through the permissions in the requested set, so block **1434** is always 'yes' and as permissions are found in the user set, a counter is incremented. At the end, if the counter does not equal the number of permissions in the user set, the DIRECTIONAL PERMISSIONS SETS RESULTS are merged with EMPTY SUBSET using the table in FIG. **20**.

The method and system of the present invention for comparing parameters within a pair of permissions is illustrated in functional flow form in FIG. **14**C. FIG. **14**C is substantially identical to FIG. **14**B except that FIG. **14**C is at the parameter level and FIG. **14**B is at the permission set level. First, a variable designated DIRECTIONAL PERMISSION COMPARISON RESULT is initialized to EMPTY (block **1450**). Next, the process attempts to retrieve a parameter from one permission and then tries to retrieve a corresponding parameter from the other permission (block **1452**). Thereafter, a test is made to determine if a parameter was retrieved from the requested permission (block **1454**). If no parameter was retrieved from the requested permission, a test is made to determine if a parameter was retrieved from the user defined permission. If no parameter was retrieved from the user defined permission, a variable designated DIRECTIONAL PARAMETER COMPARISON RESULT is set equal to EMPTY (block **1457**). If a parameter was retrieved from the user defined permissions, DIRECTIONAL PARAMETER COMPARISON RESULT is set equal to EMPTY SUBSET (block **1462**). Thereafter, the DIRECTIONAL PARAMETER COMPARISON RESULT (the New Merge Result) is combined with the DIRECTIONAL PERMISSION COMPARISON RESULT (the Previous/Accumulated Merge Result) according to the merger table illustrated in FIG. **20** (block **1458**).

If a parameter has been retrieved from the requested permission (block **1454**), a test is made to determine if a parameter has been retrieved from the user defined permission (block **1464**). If no parameter has been retrieved from the user defined permissions, DIRECTIONAL PARAMETER COMPARISON RESULT is set equal to EMPTY SUPERSET (**1466**) and combined with DIRECTIONAL PERMISSION COMPARISON RESULT (block **1458**). Then, the DIRECTIONAL PARAMETER COMPARISON RESULT (the New Merge Result) is combined with the DIRECTIONAL PERMISSION COMPARISON RESULT (the Previous/Accumulated Merge Result) according to the merger table illustrated in FIG. **20** (block **1458**).

If a parameter has been retrieved from the user defined permission (block **1464**), the parameters are compared (block **1468**) and the result retrieved as the DIRECTIONAL PARAMETER COMPARISON RESULT, which is combined with DIRECTIONAL PERMISSION COMPARISON RESULT (block **1458**), as described above.

The parameters of a permission are generally known in advance based on the type of the permission. In one embodiment of the invention, there is no way to enumerate the parameters of a permission—they are simply known, by virtue of the type of the permission. For example, product documentation such as that referred to in other parts of this specification informs the user that the FileIO permission has read, write, and delete parameters. Thus, an actual embodiment of the invention will simply linearly perform the necessary options to compare and merge each of the predefined parameters of the permission. In this example, the process to compare two FileIO permissions will first initialize a variable named RESULT to the result of comparing the read parameters of the two permissions. Next, the process

will compare the write parameters of the two permissions and merge this comparison result with RESULT (using FIG. **20**), storing the merged result into variable RESULT. Finally, the process will compare the delete parameters of the two permissions, merge this comparison result with RESULT (using FIG. **20**), and return the merged result to the calling process. Note that the process of comparing the FileIO permission conceptually follows the form of FIG. **14**C, where tests **1454** and **1464** always follow the "yes" branch, and the blocks inside the loop are copied three times, one for each parameter.

The method and system of the present invention for comparing primitives within a pair of parameters is illustrated in functional flow form in FIG. **14**D. FIG. **14**D is substantially identical to FIGS. **14**B and **14**C except that FIG. **14**D is at the primitive level, FIG. **14**C is at the permission level and FIG. **14**B is at the permission set level. First, a variable designated DIRECTIONAL PARAMETER COMPARISON RESULT is initialized to EMPTY (block **1472**). Next, the process attempts to retrieve a primitive from one parameter and then tries to retrieve a corresponding primitive from the other parameter (block **1474**). Thereafter, a test is made to determine if a primitive was retrieved from the requested parameter (block **1476**). If no primitive was retrieved from the requested parameter, a test is made to determine if a primitive was retrieved from the user defined parameter (block **1478**). If no primitive was retrieved from the user defined parameter, a variable designated DIRECTIONAL PRIMITIVE COMPARISON RESULT is set equal to EMPTY (block **1480**). If a parameter was retrieved from the user defined permissions, DIRECTIONAL PRIMITIVE COMPARISON RESULT is set equal to EMPTY SUBSET (block **1482**). Thereafter, the DIRECTIONAL PRIMITIVE COMPARISON RESULT (the New Merge Result) is combined with the DIRECTIONAL PARAMETER COMPARISON RESULT (the Previous/Accumulated Merge Result) according to the merger table illustrated in FIG. **20** (block **1484**).

If a primitive has been retrieved from the requested parameter (block **1476**), a test is made to determine if a primitive has been retrieved from the user defined parameter (block **1486**). If no primitive has been retrieved from the user defined parameters, DIRECTIONAL PRIMITIVE COMPARISON RESULT is set equal to EMPTY SUPERSET (**1488**) and the DIRECTIONAL PRIMITIVE COMPARISON RESULT (the New Merge Result) is combined with the DIRECTIONAL PARAMETER COMPARISON RESULT (the Previous/Accumulated Merge Result) according to the merger table illustrated in FIG. **20** (block **1484**), as described above. If a primitive has been retrieved from the user defined parameters, then the two primitives are compared. The method and system of the present invention for comparing primitives of like type is illustrated in FIGS. **15**A–**19**I and described below. Each primitive type returns a directional comparison result called RESULT to the parameters comparison process (FIG. **14***d*) and set equal to the DIRECTIONAL PRIMITIVES COMPARISON RESULT (block **1492**), which is then combined with the DIRECTIONAL PARAMETER COMPARISON RESULT using the merger table illustrated in FIG. **20** (block **1484**), as described above

After the DIRECTIONAL PRIMITIVE COMPARISON RESULT has been combined with the DIRECTIONAL PARAMETER COMPARISON RESULT, a test is made to determine if another primitive needs to be processed (block **1490**). If another primitive needs to be processed, the process cycles to the primitives retrieval block **1472** and the foregoing steps are repeated. After all of the primitives have been processed, the DIRECTIONAL PARAMETER COMPARISON RESULT is returned (block **1468**) to the permissions comparison process (FIG. **14**C) to be combined with the DIRECTIONAL PERMISSION COMPARISON RESULT (block **1458**).

Once the DIRECTIONAL PARAMETER COMPARISON RESULT has been combined with the DIRECTIONAL PERMISSION COMPARISON RESULT (block **1458**), a test is made to determine if another parameter needs to be processed (block **1460**). If another parameter needs to be processed, the process cycles to the parameter retrieval block **1452** and the foregoing steps are repeated. After all of the parameters have been processed, the DIRECTIONAL PERMISSION COMPARISON RESULT is returned (block **1470**) to the permissions comparison process (FIGS. **14**B) to be combined with the DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT (block **1440**).

After the DIRECTIONAL PERMISSION COMPARISON RESULT has been combined with the DIRECTIONAL PERMISSIONS SET COMPARISON RESULT (block **1440**), a test is made to determine if another permission needs to be processed (block **1442**). If another permission needs to be processed, the process cycles to the permission retrieval block **1452** and the foregoing steps are repeated. After all of the permissions have been processed, the DIRECTIONAL PERMISSIONS SET COMPARISON RESULT is returned (block **1448**) to the calling block (blocks **1336**, **1346** or **1356**) of FIGS. **13**A and B to be processed as described above.

In the processes outlined by FIGS. **14**B and FIG. **14**C, the permissions and parameters were distinct units; no two permissions or parameters have overlapping meanings. Similarly, the process described by FIG. **14**D only operates on primitive types with distinct values. There are primitive types that do not naturally follow this restriction as the permissions and their parameters do. For these more complex primitive types, a special process must be executed to account for the possibility of overlapping values. Wildcard expressions are the only primitive types used in the invention that require a special primitive list comparison process. The process for is described in detail below, starting with the discussion of FIG. **18**A. In the context of FIG. **14**D, this implies that lists of such complex types must be considered as a single macro primitive.

If the primitives to be compared (block **1468**; FIG. **14**C) are inclusive Boolean primitives, the method and system of FIG. **15**A is performed. If the inclusive Boolean primitive of the parameter in the requested permission is true (block **1510**) and the inclusive Boolean primitive of the parameter in the user defined permission is also true (block **1520**), RESULT is set to EQUAL (block **1530**) and the RESULT is returned (block **1580**).

If the inclusive Boolean primitive of the parameter in the requested permission is true (block **1510**) but the inclusive Boolean primitive of the parameter in the user defined permission is not true (block **1520**), RESULT is set to SUPERSET (block **1540**) and RESULT is returned (block **1580**). If the inclusive Boolean primitive of the parameter in the requested permission is false (block **1510**) and the inclusive Boolean primitive of the parameter in the user defined permission is true (block **1550**), RESULT is set to SUBSET (block **1560**) and RESULT is returned (block **1580**). If the inclusive Boolean primitive of the parameter in the requested permission is false (block **1510**) and the inclusive Boolean primitive of the parameter in the defined

permission is false (block **1550**), RESULT is set to EQUAL (block **1570**) and RESULT is returned (block **1580**).

By way of example only, assume that the inclusive Boolean primitive of the parameter in the requested permission is true and the inclusive Boolean primitive of the parameter in the user defined permission is false, the returned RESULT (block **1580**) is SUPERSET. Turning to FIG. **14**D, assume that the existing value of DIRECTIONAL PARAMETER COMPARISON RESULT is DISJOINT. Referencing the table in FIG. **20**, when DISJOINT (cell E**1**) is combined (block **1458**) with SUPERSET (cell A**6**) the new DIRECTIONAL PARAMETER COMPARISON RESULT is OVERLAP (cell E**6**). If there are no more primitives to compare within the like parameters (FIG. **14**D), the DIRECTIONAL PARAMETER COMPARISON RESULT of OVERLAP is returned (block **1494**; FIG. **14**D) to FIG. **14**C (block **1468**). Continuing the example, assuming the existing DIRECTIONAL PERMISSION COMPARISON RESULT is DISJOINT (cell E**1**) when OVERLAP (cell A**2**) is combined (block **1440**) with DISJOINT (cell E**1**), the value of DIRECTIONAL PERMISSION COMPARISON RESULT is OVERLAP (cell E**2**). If this is the last parameter to compare, the DIRECTIONAL PERMISSION COMPARISON RESULT of OVERLAP is returned (block **1458**; FIG. **14**C) to FIG. **14**B (block **1446**). Continuing the example, assuming the existing DIRECTIONAL PERMISSIONS SETS COMPARISON RESULT is DISJOINT (cell E**1**) when OVERLAP (cell A**2**) is combined (block **1440**) with DISJOINT (cell E**1**), the value of DIRECTIONAL PERMISSIONS SET COMPARISON RESULT is OVERLAP (cell E**2**). If there are not more permissions to compare, the DIRECTIONAL PERMISSIONS SET COMPARISON RESULT of OVERLAP is returned to the appropriate decision point (blocks **1336**, **1346** or **1356**) in the permission analysis process (FIGS. **13**A–C).

If, for instance, OVERLAP is returned to the query decision block **1346**, the query decision result would be yes and process would cycle to the query user display block **1348**, which, as noted above, would cause a dialog to be displayed to the user requesting instructions on whether to approve or disapprove the permissions. Depending on the action taken, processing would continue as is described above with reference to FIGS. **13**A–C.

The manner of processing the DIRECTIONAL PRIMITIVE COMPARISON RESULT returned by the primitive comparison step (block **1468**) is the same regardless of the type of primitive that is being compared. The following discussion only addresses how the remaining primitive types are compared with the understanding that once a RESULT is returned and assigned to the DIRECTIONAL PRIMITIVE COMPARISON RESULT, further processing occurs in the manner generally outlined above.

Comparison of exclusive Boolean primitives is illustrated in FIG. **15**B. An exclusive Boolean primitive represents an action that is either allowed or disallowed, in contrast to an inclusive Boolean primitive wherein a "false" means that "part" of the action is allowed but to a lesser extent than a value of "true". If the exclusive Boolean primitive of the parameter in the requested permission is true (block **15100**) and the exclusive Boolean primitive of the parameter in the user defined permission is true (block **15110**), RESULT is set to EQUAL (block **15120**) and the RESULT is returned (block **15180**). If the exclusive Boolean primitive of the parameter in the requested permission is set to true (block **15100**) but the exclusive Boolean primitive of the parameter in the user defined permission is false (block **15110**), RESULT is set to EMPTY SUPERSET (block **15130**) and RESULT is returned (block **15180**). If the exclusive Boolean primitive of the parameter in the requested permission is false (block **15100**), and the exclusive Boolean primitive of the parameter in the user defined permission is true (block **15140**) RESULT is set to EMPTY SUBSET (block **15150**) and RESULT is returned (block **15180**). If the exclusive Boolean primitive of the parameter in the requested permission is false (block **15100**) and the exclusive Boolean primitive of the parameter in the user defined permission is false (block **15140**), RESULT is set to EMPTY (block **15160**) and RESULT is returned (block **15180**).

In an actual embodiment, the primitive type of a permission parameter is generally known, so the process for comparing the parameters is reduced to a linear process that simply compares the underlying primitive types. For example, the UserFileIOPermission consists of two parameters, each of which is an exclusive Boolean primitive type. The process for comparing the UserFileIOPermission is to first initialize a variable named RESULT to the result of comparing the read access exclusive Booleans of the two permissions. Next, the write access exclusive Booleans of the two permissions are compared, and the result is combined with RESULT using FIG. **20**. Finally, RESULT is returned to the permission set comparison process (block **1440**). Thus, because the types and quantities of the parameters are known, several steps from the permission comparison process are eliminated in the actual implementation, but still follow the spirit of the process.

The method and system for comparing primitives consisting of ordered or unordered lists to produce a directional set relationship is illustrated in FIG. **16**A. FIG. **16**A is substantially similar to FIGS. **14**B–D except that FIG. **16**A is at the primitive level and has the additional step of merging extra data associated with the primitives being compared, to allow for nested structures. Extra data is specified, for example, in the Net permission. The Net permission has a primary list of integers, and each integer in the primary list can have a secondary list of integers associated with it. The primary list is the list of network addresses and the secondary list is the ports associated with the network addresses. If a secondary list is not specified, this implicitly means that all ports are specified for that network address.

First, a variable designated RESULT is initialized to EMPTY (block **1610**). A test is made to determine if another element needs to be processed (block **1615**) in either list. If another element needs to be processed, an attempt is made to retrieve an element from one list and then an attempt is made to retrieve a corresponding element from the other list (block **1620**). Thereafter, a test is made to determine if an element was retrieved from the requested list (block **1625**).

If an element has been retrieved from the requested list (block **1625**), a test is made to determine if an element has been retrieved from the user defined list (block **1630**). If an element has been retrieved from the user defined list, a variable designated LIST RESULT is set to EQUAL (block **1635**). Next, a test is made to determine if the list elements have extra primitives associated with them (block **1650**). If so, the associated primitives are compared (block **1655**) and a variable designated EXTRA DATA COMPARISON RESULT is set to the result of the comparison of the extra data primitives. The EXTRA DATA COMPARISON RESULT is merged with LIST RESULT using the merger table illustrated in FIG. **16**B to set a new value for the LIST RESULT.

If no element has been retrieved from the user defined list (block **1630**), LIST RESULT is set equal to EMPTY

SUPERSET (**1640**). If no element was retrieved from the requested list (block **1625**), then it must have been retrieved from the user list, and LIST RESULT is set equal to EMPTY SUBSET (block **1645**).

After the LIST RESULT has been set (blocks **1640**, **1640** or **1660**), the LIST RESULT is combined with the RESULT according to the merger table illustrated in FIG. **20** (block **1665**). The process cycles to the remaining elements test block **1615** and the foregoing steps are repeated. After all of the elements have been processed, the RESULT is returned to the primitive comparison process (FIG. **14**D) as the DIRECTIONAL PRIMITIVE COMPARISON RESULT (block **1492**).

The comparison of numerical limits primitives is illustrated in FIG. **17**. If the requested permission limit and the user defined permission limit both equal zero (block **1710**), RESULT is set to EMPTY (block **1712**) and returned (block **1728**). If the requested permission limit and user defined permission limit do not equal zero (block **1710**), a test is made (block **1714**) to determine if the user defined permission limit equals zero. If the user defined permission limit equals zero, RESULT is set to EMPTY SUPERSET (block **1716**) and returned (block **1728**). If the user defined permission limit is not zero, a test is made (block **1718**) to determine if the requested permission limit equals zero. If the requested permission limit equals zero, RESULT is set to EMPTY SUBSET (block **1720**) and returned (block **1728**). If the requested permission limit is not equal to zero, a test is made (block **1722**) to determine whether the requested permission limit is greater than the user defined permission limit. If the requested permission limit is greater than the user defined permission limit, RESULT is set to SUPERSET (block **1724**) and returned (block **1728**). If the requested permission limit is not greater than the user defined permission limit, RESULT is set to SUBSET (block **1726**) and returned (block **1728**).

The comparison of regular expressions and the assignment of a directional primitive RESULT is shown in FIGS. **18**A–AA. A regular expression is a string of characters that provide a reference to an item or a set of items. For instance, a regular expression that references a file may be a character string such as "mydoc.doc". A regular expression that references a set of items contains one or more "wildcard" characters that represent one or many other characters. For instance, the regular expression "*.doc" is a regular expression that includes the wildcard character "*". In computer systems that use the Microsoft Windows operating system, the "*" character in a regular expression indicates that one or more characters may be substituted for the "*" character at the "*" wildcard's position in the regular expression. The regular expression, therefore, represents all regular expressions that end in the string ".doc", including "mydoc.doc". The Microsoft Windows operating system also recognizes the "?" wildcard character to represent that any single character can be substituted in the regular expression for the "?" character. An example of a regular expression that includes the "?" wildcard character is "???.doc", which indicates any regular expression that has three characters immediately followed by the ".doc" character string. Using this regular expression, a file named "aaa.doc" or "abc.doc" would be considered by the operating system to be an equivalent regular expression, while the filename "mydoc.doc" would not because it has more than three characters that precede the ".doc" character string.

Comparing Regular Expressions

The comparison of regular expressions to return a directional primitive comparison PERMISSION RESULT is illustrated in FIGS. **18**A–**18**AA. FIG. **18**A illustrates a method and system formed in accordance with this invention for comparing a first expression ("EXPRESSION1") and a second expression ("EXPRESSION2") in a functional flow diagram form. EXPRESSION1 is a regular expression that defines a parameter from the requested permission and EXPRESSION2 is a regular expression defines a corresponding parameter from the user defined permission.

To facilitate the comparison process, the expressions will be separated into "component groups". A single component is either a character, hereafter referred to as a "MATCH" component type; a wildcard "?", referred to as a "SKIP" type; or a wildcard "*", referred to as a "CONSUME" type. A component group is a contiguous region of like components.

A brief overview of the drawings associated with this process:

FIG. **18**A describes at the highest level the process of comparing two expressions and initializes various variables used through the process.

FIG. **18**B begins the process of "normalizing" the expression and identifying component groups.

FIG. **18**C, referred to by FIG. **18**B, appends the appropriate codes to an array describing a component group.

FIG. **18**D is a recursive process that compares sequences of component groups in the expressions, when the current state of the comparison process is at the start of component groups in each expression.

FIG. **18**E is a recursive process that attempts to locate compatible components in EXPRESSION2, given a whole or partial component group from EXPRESSION1. The component group from EXPRESSION1 will be a SKIP group or a MATCH group, and a variable named CNS**1** will be set to "true" if the component group was preceded by a CONSUME group.

FIG. **18**F is a recursive process that attempts to match whole or partial SKIP or MATCH groups from EXPRESSION1 against whole or partial SKIP or MATCH groups in EXPRESSION2; each group may or may not be preceded by a CONSUME group, as will be indicated by the CNS**1** and CNS**2** variables, respectively.

FIG. **18**G is a process used to reverse the comparison orientation.

FIGS. **18**H–J describe a recursive process that attempts to match whole or partial SKIP or MATCH groups from EXPRESSION1 against whole or partial SKIP or MATCH groups in EXPRESSION2. If the group from EXPRESSION1 is preceded by a CONSUME group, indicated by the CNS**1** variable, components of the group from EXPRESSION2 will be skipped and the remaining components from EXPRESSION1 and EXPRESSION2 will be compared from the skipped location.

Referring to FIG. **18**A, EXPRESSION1 is first normalized (block **18A010**). This is accomplished by a NormalizeExpressionAndSeparateComponentGroups process illustrated in flowchart form in FIG. **18**B. This process identifies the locations of "component groups" in the expression to later facilitate the comparison process. A single component is either a character, hereafter referred to as a "MATCH" component type; a wildcard "?", referred to as a "SKIP" type; or a wildcard "*", referred to as a "CONSUME" type. The process identifies the locations of contiguous regions of like component types and their lengths. It also "normalized" the input expression by removing degenerate cases such as consecutive CONSUME components, ensuring that SKIP

groups that are adjacent to CONSUME groups are both preceded and followed by CONSUME groups, and reducing CONSUME-SKIP-CONSUME-SKIP-CONSUME patterns to simpler CONSUME-SKIP-CONSUME patterns.

As shown in FIG. **18**B, the first step of the NormalizeExpressionAndSeparateComponentGroups process is to set a variable designated LASTTYPE, which is the type of the last component that was discovered in the input expression, to NONE. Next a variable designated PTRLIST is set to {}, which means an empty array (block **18**B**014**). The input expression is the value of EXPRESSION1 passed from block **18**A**010** in FIG. **18**A. Next, a variable designated GROUPSTART, which tracks the start of the current component group being formed, is initialized to zero (block **18**B**016**). Then, a variable designated I, which tracks the current location of the processing of the input expression, is initialized to zero (block **18**B**018**).

The PTRLIST variable is an array that accumulates codes describing the locations of component groups in the input expression. The codes stored in the array depend on the component group type. For MATCH groups, this consists of a starting index into the original expression and a count of the number of characters to match. For SKIP groups, this consists of the number of characters to skip. For CONSUME groups, no additional information is needed. In the current embodiment, the codes are stored as variable-length sequences of integers. The first integer identifies the component group type. Because indices into the input expression will always be positive, a positive integer identifies a MATCH group, and also indicates the starting position of the group. Negative values for the first integer identity SKIP and CONSUME groups. For MATCH and SKIP groups, the number of characters to match or skip is indicated by the second integer. Thus, MATCH and SKIP groups are encoded as two integers; CONSUME groups are encoded as a single integer.

PTRLIST is an array that accumulates codes describing the locations of component groups in the input expression. The codes stored in the array depend on the component group type. For MATCH groups, this consists of a starting index into the original expression and a count of the number of characters to match. For SKIP groups, this consists of the number of characters to skip. For CONSUME groups, no additional information is needed. In the current embodiment, the codes are stored as variable-length sequences of integers. The first integer identifies the component group type. Because indices into the input expression will always be positive, a positive integer identifies a MATCH group, and also indicates the starting position of the group. Negative values for the first integer identify SKIP and CONSUME groups. For MATCH and SKIP groups, the number of characters to match or skip is indicated by the second integer. Thus, MATCH and SKIP groups are encoded as two integers; CONSUME groups are encoded as a single integer.

The NormalizeExpressionAndSeparateComponentGroups method processes each component in the input expression in a loop that begins with a test (block **18**B**020**) that determines if I is at the end of the input expression. If I is not at the end of the input expression, the next component in the input string is retrieved for processing and I is incremented (block **18**B**022**). The component is then analyzed and a variable designated TYPE is set to a value that is dependent on the nature of the component (block **18**B**024**). If the component is a character, TYPE is set to MATCH. If the component is the wildcard "*", TYPE is set to CONSUME. If the component is the wildcard "?", TYPE is set to SKIP. Next, a test is made (block **18**B**026**) to determine if the value of TYPE is different than the value of LASTTYPE. A difference indicates a new component group is beginning. If the answer to the test (block **18**B**026**) is yes, codes describing the last component group are added to PTRLIST (such as pointer, length, type, etc.) and a new component group is then started and described (block **18**B**028**) using the DescribeComponentGroup process illustrated in FIG. **18**C.

The DescribeComponentGroup process shown in FIG. **18**C begins with a test block (block **18**C**030**) that determines whether LASTTYPE equals SKIP. If LASTTYPE equals SKIP, a test is made (block **18**C**032**) to determine whether the last two component groups added to the PTRLIST array were a SKIP group followed by a CONSUME group. This checks for a "*?*?" variant, which if found, is collapsed into a "*??*" component group by extending the previous SKIP group. This is accomplished by adding the components defined by the length of the group (which is found by subtracting the current position of the processing minus where the group started) to the existing SKIP group already in the PTRLIST array (block **18**C**034**). At this point, the DescribeComponentGroup method (FIG. **18**C) is complete (block **18**C**046**).

If the last two components added to the PTRLIST array were not a SKIP group followed by CONSUME group (block **18**C**032**), an adjustment must be made to the PTRLIST array so that SKIP groups adjacent to a CONSUME group are both preceded and followed by a CONSUME group. This is accomplished by first making a test (block **18**C**036**) to determine if the current group type is CONSUME and if the PTRLIST array is EMPTY or does not end with a CONSUME group. If the answer is yes, a CONSUME group is added to the PTRLIST array (block **18**C**038**) before the SKIP group comprising the current group is added to the PTRLIST array (block **18**C**040**) (with a group size equal to the difference between the current value of I and current value of GROUPSTART). If a CONSUME group is already present in the PTRLIST array (block **18**C**036**), the SKIP group is added to the PTRLIST array (block **18**C**040**) without adding an intervening CONSUME group to the PTRLIST array (block **18**C**038**).

Next, a test is made (block **18**C**042**) to determine if the type of the current group is not CONSUME and if a CONSUME group preceded the SKIP group just created. If the answer is yes, a CONSUME group is added to the PTRLIST array (block **18**C**044**) and the process is done (block **18**C**046**). If the current group is CONSUME and a CONSUME group preceded the SKIP group just created, the CONSUME groups and SKIP groups are balanced and the process in FIG. **18**C is done (block **18**C**046**).

If LASTTYPE was not a SKIP group (block **18**C**030**), a test (block **18**C**048**) is made to determine if LASTTYPE is a CONSUME group. If LASTTYPE is a CONSUME group, a test (block **18**C**050**) is made to determine if PTRLIST is EMPTY or if the last component group added to PTRLIST was not a CONSUME group. If the answer is no, the process is done (block **18**C**054**). If the answer is yes, a CONSUME group is added to the PTRLIST (block **18**C**052**) and the process is done (block **18**C**054**).

If LASTTYPE was not a CONSUME group (block **18**C**048**), a test (block **18**C**056**) is made to determine if I is greater than GROUPSTART. If I is not greater than GROUPSTART, the process is done (block **18**C**060**). If I is greater than GROUPSTART LAST GROUP is an EMPTY string. If I is greater than GROUPSTART, a MATCH group is appended to the PTRLIST with a length of I minus

GROUPSTART (block **18C058**) and the process is done (block **18C060**). When the process illustrated in FIG. **18C** is done, processing returns to FIG. **18B** (block **18B028**). The returned PTRLIST is assigned to the variable PTRLIST**1** which is array of the component group descriptor codes for EXPRESSION1.

Returning to FIG. **18B**, after the DescribeComponentGroup process FIG. **18C**) is complete, the value of GROUPSTART is set equal to the current value of I (block **18B062**), which is the current location of the start of the next component group. Next, LASTTYPE is set to TYPE (block **18B064**). This is done to remember the type of the current component as the type of the next component. After the foregoing steps are completed, or if TYPE is not different from LASTTYPE, a test (block **18B066**) is made to determine if TYPE is END. If not, the process is repeated beginning with decision block **18B020**. If TYPE variable is END, a list of the component group descriptor codes (PTRLIST) is returned (block **18B068**) to the Compare Expression process (block **18A010**).

Returning to FIG. **18A**, EXPRESSION2 is normalized (block **18A070**) using the NormalizeExpressionAndSeparateGroup process (FIG. **18B**) in the same manner as just described for the normalization (block **18A010**) of EXPRESSION1. The returned PTRLIST is assigned to the variable PTRLIST**2**, which is an array of the component group descriptor codes for EXPRESSION2.

Once EXPRESSION1 has been normalized into PTRLIST**1** and EXPRESSION2 has been normalized into PTRLIST**2**, the component groups are compared (block **18A072**) using a CompareComponentGroups process shown in flowchart form in FIG. **18D**, with the variables CNS**1** initialized to false, CNS**2** initialized to false and RESULT initialized to EQUAL and pointers P**1** and P**2** initialized to zero.

Referring to FIG. **18D**, the CompareComponentGroups flowchart begins with a test (block **18D074**) to determine whether P**1** is at the end of PTRLIST**1**. If not, the next component group is retrieved using a descriptor from PTRLIST**1** (block **18D076**) using the value of P**1** as the index. P**1** is then incremented (block **18D078**). The component group descriptor just retrieved (block **18D076**) is checked (block **18D080**) to see if the group is a CONSUME group. If the group is a CONSUME group, CNS**1** is set to true and the process cycles to the P**1** at the end of PTRLIST**1** test (block **18D074**). If the group is not a CONSUME group (block **18D080**), then N**1** is set to the size of the group stored with the group descriptor of the component group being processed (block **18D082**). The component group descriptor is then tested (block **18D084**) to see if the group is a SKIP group. If the group is a SKIP group, a variable designated ANY**1** is set to true (block **18D086**) and a variable designated I**1** is set equal to zero (block **18D088**). If the group is not a SKIP group (block **18D084**), ANY**1** is set equal to false (block **18D090**) and I**1** is set equal to the starting character position of the group that is stored in the PTRLIST**1** associated with the component group descriptor being processed (block **18D092**).

Next, the CompareComponentGroups process attempts to find (block **18D094**) a matching group in PTRLIST**2** using a FindComponents process illustrated in flowchart form in FIG. **18E**. At this point, P**1** is pointing to the group in PTRLIST**1** that follows the group that will be looked for in PTRLIST**2**. As described below, this relation is used later in the process for the intersection notations.

Turning to FIG. **18E**, the FindComponents begins with a test (block **18E096**) to determine if P**2** is at the end of PTRLIST**2**. If P**2** is not at the end of PTRLIST**2**, the next component group descriptor from the PTRLIST**2** is retrieved at the index P**2** (block **18E098**). P**2** is then incremented (block **18E100**).

Next, a test (block **18E102**) is made to determine whether the group is a CONSUME group. If the group is a CONSUME group, CNS**2** is set to true (block **18E104**) and the process cycles to test whether P**2** is at the end of PTRLIST**2** (block **18E096**).

If the group is not a CONSUME group (block **18E102**), a variable designated N**2** is set (block **18E106**) to the size of the group that is stored with the component group descriptor in PTRLIST**2** at index P**2**. Then a test is made to determine if the component group is a SKIP group (block **18E108**). If the group is a SKIP group, a variable designated ANY**2** is set to true (block **18E110**) and a variable designated I**2** is set equal to zero (block **18E112**). If the group is not a SKIP group, ANY**2** is set to false (block **18E114**) and I**2** is set to the starting character position of the group that is stored with the component group descriptor in the PTRLIST**2** (block **18E116**).

At this point, both P**1** and P**2** reference the groups following the groups from PTRLIST**1** and PTRLIST**2**. As discussed below, these values for P**1** and P**2** are used later in the method for the intersection notations. For CHARACTER groups, Ix points to the start of the characters to compare in EXPRx, and Nx is the number of characters to align. For SKIP groups, Ix is zero and Nx is the number of "?" characters. After the ANY**2** variable has been set (block **18E110** or block **18E114**) and the I**2** variable has been set (block **18E112** or block **18E116**) the component groups are compared (block **18E118**) using an AlignComponents process illustrated in flowchart form in FIG. **18F**.

In the first step of the AlignComponents process a test is made to determine if CNS**2** is true (block **18F122**). If CNS**2** is true, the comparison orientation is reversed using a SwapExpressions process illustrated in flowchart form in FIG. **18G**.

The SwapExpressions process temporarily switches EXPRESSION1 and its variables with EXPRESSION2 and its variables. For example, if comparing "a*" against "*?*" after performing the SwapExpressions procedure the comparison will be "*?*" against "a*." In other words, this procedure simply swaps the state of all comparisons/intersection variables. This can be accomplished in many computer languages by simply reversing the order of the parameters passed to a function.

Turning to FIG. **18G**, first EXPR**1** and EXPR**2** are swapped (block **18G126**), next the PTRLIST**1** and PTRLIST**2** arrays are swapped (block **18G128**), then the values of I**1** and I**2** are swapped (block **18G130**), next the values of P**1** and P**2** are swapped (block **18G132**), then the values of CNS**1** and CNS**2** are swapped (block **18G134**), next the values of ANY**1** and ANY**2** are swapped (block **18G136**). Thereafter, the value of the RESULT variable is then inverted (block **18G138**) using Table **18-5** illustrated in FIG. **18Y**. Finally, the value of the variable SWAPPED is negated and the process returns (block **18G144**) to the AlignComponents process (block **18F124**, FIG. **18F**). In some embodiments of this invention, not all of the variables swapped by the SwapExpressions process shown in FIG. **18F** will be necessary. For brevity, a reference to this procedure simply indicates that the comparison/intersection orientation should be reversed, and the most efficient implementation of this procedure will depend largely on the language used and other implementation-specific details. In one actual embodi-

ment of this invention, the SwapExpressions process is responsible only for swapping EXPR1/EXPR2, PTRLIST1/PTRLIST2, and negating SWAPPED. All other states are swapped by reversing parameters to procedures that follow from the foregoing generic description.

Returning to FIG. **18**F, after the SwapExpressions process is finished (block **18G144**), the AlignComponents process recursively finds (block **18F146**) components from EXPRESSION1 in EXPRESSION2 using a ShiftComponents process illustrated in flowchart form in FIG. **18**H. While this is done, the values of the variables CNS**1**, ANY**1**, P**1**, N**1**, CNS**2**, ANY**2**, P**2**, I**2**, N**2**, and RESULT are preserved. In languages that use pass-by-value function calling semantics, these variables may be preserved simply as part of the mechanics of calling a function, so that no explicit action to preserve these variables is required. The value of a variable designated SHIFTRESULT is assigned the value of the variable RESULT produced by the ShiftComponents process.

As shown in FIG. **18**I, if a ShiftComponents process is being used for intersecting, the value of a variable designated CURISECTPTR is preserved during the operation (block **18H148**). Next, CNSRESULT is set to DISJOINT (block **18H150**). Next, the ShiftComponents process adjusts RESULT if one component group is preceded by a CONSUME group but the other group is not so preceded. This is accomplished starting with a test (block **18H152**) to determine if CNS**1** is equal to CNS**2**. If CNS**1** equals CNS**2** (block **18H152**), no adjustment to RESULT is necessary. If CNS**1** is not equal to CNS**2**, then a test is made to determine if CNS**1** is true (block **18H154**). If CNS**1** is true, RESULT is adjusted (block **18H156**) using Table **18-3** FIG. **18**X). If CNS**1** determines is not true (block **18H154**), RESULT is adjusted (block **18H160**) using Table **18-4** (FIG. **18**X). After adjustment, or if no adjustment is required, a variable designated BIASRESULT is set equal to RESULT (block **18H164**). BIASRESULT is used to represent comparing "?" groups against character groups, and is adjusted if consumption occurs.

After BIASRESULT is set equal to RESULT (block **18H164**), a test is made to determine if ANY**1** is equal to ANY**2** (block **18H166**). If ANY**1** is not equal to ANY**2**, a test is made to determine if ANY**1** is true (block **18H168**). If ANY**1** is true, BIASRESULT is adjusted (block **18H170**) using Table **18-3** (FIG. **18**X). If ANY**1** is not true (block **18H168**), the BIASRESULT is adjusted (block **18H172**) using Table **18-4** (FIG. **18**X).

If ANY**1** equals ANY**2** or after BIASRESULT is adjusted, a test is made (block **18H174**) to determine if both CNS**1** and ANY**1** are true. If both CNS**1** and ANY**1** are true, a variable designated BIASCNS**1** is set to true (block **18H176**). If either the value of CNS**1** or ANY**1** is not true, BIASCNS**1** is set to false (block **18H178**). The state of BIASCNS**1** indicates whether or not the group from PTRLIST**1** can continue to be considered a consumer after a match is found. A consumer consumes groups.

After the value of BIASCNS**1** is set (block **18H176** or block **18H178**), a test is made to determine if CNS**2** and ANY**2** are both true (block **18H180**). If both CNS**2** and ANY**2** are true, a variable designated BIASCNS**2** is set to true (block **18H182**). If either the value of CNS**2** or ANY**2** is not true, BIASCNS**2** is set to false (block **18H184**). Like BIASCNS**1**, the state of BIASCNS**2** indicates whether or not the group from PTRLIST**2** can continue to be considered a consumer after a match is found.

After BIASCNS**2** is set (block **18H182** or block **18H184**), a ShiftComponents**2** process shown in flowchart form in FIG. **18**I is used (block **18H186**) to attempt to find matching components in the expressions being compared.

The ShiftComponents**2** process illustrated in FIG. **18**I begins by comparing characters at the current location (block **18I188**) using a CompareSingleComponents process illustrated in flowchart form in FIG. **18**J.

Referring to FIG. **18**J, the CompareSingleComponents process begins by assigning the value of a variable designated TOMATCH to be the lesser of the value of N**1** or N**2** (block **18J190**). As noted above, Nx is the number of characters to align. Thus, N**1** is the number of PTRLIST**1** characters and N**2** is the number of PTRLIST**2** characters. Next, ANY**1** is checked to determine if it is true (block **18J192**). If ANY**1** is not true, ANY**2** is checked to determine if it is true (block **18J194**). If neither ANY**1** or ANY**2** is true, the number of characters in EXPRESSION1 represented by the variable TOMATCH, starting at the position equal to the value of I**1**, are compared to the characters in EXPRESSION2, starting at position indicated by the variable I**2** (block **18J224**). Then a test is made to determine if the character ranges are equal (block **18J226**). If the character ranges are not equal, a "found indefinite result" message is returned (block **18J228**) to the comparison call (block **18I188**) of the ShiftComponents**2** process illustrated in FIG. **18**I.

At this point, if all of the characters or "?" characters of both groups have been exhausted, the method continues comparing the following groups in PTRLIST**1** and PTRLIST**2** using the current values of P**1** and P**2**, respectively, as their starting points. In this regard, if the character ranges are equal (block **18J226**) or if ANY**1** or ANY**2** are true (block **18J192** or block **18J194**), a test is made to determine if N**1** is equal to N**2** (block **18J196**).

If N**1** is equal to N**2** (block **18J196**), the remaining component groups are recursively compared using the CompareComponentsGroups process illustrated in flowchart form in FIG. **18**D, and discussed in detail in other portions of this application. While the CompareComponentsGroups process is being executed, the values of CNS**1**, ANY**1**, P**1**, N**1**, CNS**2**, ANY**2**, P**2**, I**2**, N**2**, and RESULT are preserved. Further the value of CNS**1** is temporarily assigned the value of BIASCNS**1**, the value of CNS**2** is temporarily assigned the value of BIASCNS**2**, and the value of RESULT is temporarily assigned the value of BIASRESULT. Also, the recursive comparison (block **18J198**) sets the value of a variable designated MATCHRESULT to the result returned by the CompareComponentsGroups process.

If N**1** is not equal to N**2** (block **18J196**), then the CompareSingleComponents process checks for characters or "?" characters remaining in one of the groups and then attempts to find them in the opposite expression. This part of the process is commenced by testing TOMATCH to determine if it is equal to N**1** (block **18J200**). If TOMATCH is not equal to N**1**, the CompareSingleComponents process recursively finds the remaining components from EXPRESSION1 in EXPRESSION2 using the FindComponents process illustrated in FIG. **18**E and discussed in detail in other portions of this specification (block **18J202**).

While the FindComponents process recursively finds the remaining components from EXPRESSION1 and EXPRESSION2 (block **18J202**), the values of CNS**1**, ANY**1**, P**1**, N**1**, CNS**2**, ANY**2**, P**2**, I**2**, N**2**, and RESULT are preserved. Further, during the recursive find (block **18J202**), the process temporarily uses the value of BIASCNS**1** for CNS**1**, BIASCNS**2** for CNS**2**, I**1** plus the value of TOMATCH for I**1**, N**1** minus the value of TOMATCH for N**1**, and the value

of BIASRESULT for RESULT. MATCHRESULT is set to the value of the result returned by the FindComponents process.

If TOMATCH is equal to the value of **N1** (block **18J200**), the current state is swapped (block **18J204**) using the SwapExpressions process illustrated in FIG. **18**G and described above. After the swap, BIASRESULT is inverted (block **18J206**) using Table **18-5** shown in FIG. **18**Y. Next, the CompareSingleComponents process recursively finds (block **18J208**) the remaining components from EXPRESSION1 in the EXPRESSION2 using the FindComponents process illustrated in FIG. **18**E and discussed in detail in other portions of this specification. While the FindComponents process is being executed, the values of CNS**1**, ANY**1**, **P1**, **N1**, CNS**2**, ANY**2**, **P2**, **I2**, **N2**, and RESULT are preserved. Also during the execution of the FindComponents process, the method temporarily uses the value of BIASCNS**1** for CNS**2**, BIASCNS**2** for CNS**1**, **I1** plus the value of TOMATCH for **I1**, the value of **N1** minus the value of TOMATCH for **N1**, and BIASRESULT for RESULT. The temporary assignment of BIASCNS**1** and BIASCNS**2** is done because these variables are not swapped by the SwapExpressions procedure illustrated in FIG. **18**G. MATCHRESULT is set to the value of the result returned by the FindComponents process.

After the value of MATCHRESULT is determined, the value is inverted (block **18J210**) using Table **18-5** shown in FIG. **18**Y. The CompareSingle-Components process then inverts (block **18J212**) BIASRESULT using Table **18-5**. Then the current states are swapped back (block **18J214**) using the SwapExpressions process illustrated in FIG. **18**G and described above.

After the value of MATCHRESULT has been found in the manner described above, the CompareSingleComponents process checks (block **18J216**) to determine if MATCHRESULT is SUBSET, SUPERSET, or EQUAL. If MATCHRESULT is SUBSET, SUPERSET, or EQUAL, a "found definite result" message is returned (block **18J218**) to the comparison call (block **18I188**) in the ShiftComponents**2** process illustrated in FIG. **18**I. If MATCHRESULT is not SUBSET, SUPERSET, or EQUAL, MATCHRESULT is tested (block **18J220**) to determine if it is not DISJOINT. If MATCHRESULT is not DISJOINT, CNSRESULT is set to the value of MATCHRESULT (block **18J222**). If the value MATCHRESULT is not DISJOINT, or after CNSRESULT has been set to MATCHRESULT, the message "found indefinite result" is returned (block **18J224**) to the comparison call (block **18I188**) in the ShiftComponents**2** process illustrated in FIG. **18**I.

Returning to FIG. **18**I, after a result is returned from the CompareSingleComponents process (FIG. **18**J), the result is tested (block **18I230**) to determine if a definite result was found. If a definite result was found, MATCHRESULT is returned from the ShiftComponent**2** process to the ShiftComponents process illustrated in FIG. **18**H (block **18H186**).

If a definite result was not returned by the CompareSingleComponents process, CNS**1** is tested (block **18I234**) to determine if it is true. At this point, if the group from PTRLIST**1** is not preceded by a consumer, nothing in the group from PTRLIST**2** can be skipped. If the group from PTRLIST**2** is a consumer, this will be dealt with by the AlignComponents process illustrated in FIG. **18**F. If the test (block **18I234**) determines that CNS is not true, CNSRESULT is returned (block **18I236**) to the ShiftComponents process illustrated in FIG. **18**H (block **18H186**).

If CNS**1** is true (block **18I234**), the procedure next skips a single character or a "?" in the group from PTRLIST**2** and tries to find a match in PTRLIST**1**. Skipping starts by incrementing **I2** and decrementing **N2** (block **18I238**). If the ShiftComponents2 process is performing an intersection (discussed below), the NoteConsuming process illustrated in FIG. **18**L is executed (block **18I240**).

The NoteConsuming process illustrated in FIG. **18**L begins by setting CNSSTART equal to **P1** (block **18L242**). **P1** points to the group that follows the group that is actually doing the consumption. The NoteConsuming process (FIG. **18**L) is never executed for trailing CONSUME groups. Thus, the execution of the NoteConsuming process means that there is a CONSUME group before the group that is just before the group indicated by **P1**. In other words, the last two group descriptors before **P1** are either CONSUME and MATCH (characters x-y) or CONSUME and SKIP (n characters).

The next step in the NoteConsuming process is a test to determine if the variable SWAPPED (which indicates the existence of a swap) is true (block **18L244**). In this regard, the value for **P1** stored in the notation is encoded to indicate which expression was doing the consumption. If the original EXPRESSION2 was doing the consumption, then SWAPPED will be true, since after the swap, the original EXPRESSION2 is now EXPRESSION1. If SWAPPED is true, the length of PRTLIST**2** is added to CNSSTART (block **18L246**). In this regard, because **P1** points after the group doing the consumption, it is possible to note a value of **P1** that is the length of PTRLIST**1**. It is impossible to have a value of **P1** equal to zero, so there is no need to add one or otherwise adjust this value to distinguish between the end of PTRLIST**1** and the beginning of PTRLIST**2**.

If SWAPPED is not true (block **18L244**), or after CNSSTART has been adjusted (block **18L246**), a test is made (block **18L248**) to determine if an intersection notation buffer has been created. If an intersection notation buffer has not been created, an intersection notation buffer is created (block **18L250**). When creating the intersection notation buffer, the upper bound of the number of notations needed will be the sum of the maximum number of possible consumers in each expression, minus any trailing consumers. In the worst case, consumers are defined by a pattern of three characters, of the form "*?a", which results in two consumers. This means that the worst case number of characters per consumer is 1.5. Therefore, the worst case estimate of the number of notations is the sum of the expression character lengthy divided by 1.5. In an actual embodiment of the present invention, three integers are required for each notation, so twice the sum of the expression character lengths is used in the creation of the intersection notation buffer.

If an intersection buffer has been created (block **18L248**) or after an intersection notation buffer has been created (block **18L250**), a test is made (block **18L252**) to determine whether the value of CNSSTART matches the value of CNSSTART of the last notation added. If the value of CNSSTART matches the value of CNSSTART of the last notation added, a variable designated CURISECTPTR is set or moved to the start of the last notation (block **18L254**). This is done to extend the existing notation if the same group is consuming two consecutive target groups.

If the value of CNSSTART does not match the CNSSTART of the last notation added (block **18L252**), or after CURISECTPTR has been adjusted (block **18L254**), a notation with the values of CNSSTART, **P2** and **I2** is added to the

notation buffer at the index CURISECTPTR (block **18L256**). In an actual implementation of the invention, these are stored as three consecutive integers. CURISECTPTR is then incremented (block **18L258**) and the process returns to the ShiftComponents**2** procedure illustrated in FIG. **18I** (block **18I240**).

Returning to FIG. **18I**, after the NoteConsuming process is ended, a test is made (block **18I260**) to determine if **N2** equals 0. If **N2** is 0, CNS**2** is set to BLASCNS**2** (block **18I262**). The step is not strictly necessary because conceptually, if the group being consumed is a SKIP group and was preceded by a CONSUME group, then the next group is implicitly preceded by a CONSUME group. An explicit CONSUME group has already been inserted by the NormalizeExpressionAndSeparateComponentGroups process illustrated in FIG. **18**B for this case.

The ShiftComponents**2** process (FIG. **18**I) next determines if the entire group from the PTRLIST**2** can be skipped by recursively finding components from EXPRESSION1 in EXPRESSION2 using the FindComponents process illustrated in FIG. **18**E and discussed in detail in other portions of this specification. The result of the FindComponents process is returned as FINDRESULT (block **18I264**). Next, a test is made to determine if FINDRESULT is anything other than DISJOINT (block **18I268**). If FINDRESULT is not DISJOINT, CNSRESULT is set equal to FINDRESULT (block **18I270**). CNSRESULT is returned to the ShiftComponents process (block **18H186**) shown in FIG. **18**H.

If **N2** is not equal to zero (block **18I260**), BIASRESULT is adjusted using Table **18-3** (FIG. **18**X). Thereafter, the ShiftComponents**2** process cycles to the compare characters at the current location step (block **18I188**).

Returning to the ShiftComponents process illustrated in FIG. **18**H, if the ShiftComponents process was being used while intersecting, the saved value of CURISECTPTR is restored (block **18H274**) and, then, the value of CNSRESULT is returned (block **18H276**) to the AlignComponents process illustrated in FIG. **18**F (block **18F146**).

Returning to FIG. **18**F, the orientation of the expressions being compared that was reversed (block **18F124**) is returned (block **18F276**) to its original orientation using the SwapExpressions process illustrated in FIG. **18**G, which is discussed in detail above. Next, the value of SHIFTRESULT is then checked (block **18F278**) to determine if it equals SUBSET, SUPERSET or EQUAL. If the SHIFTRESULT is equal to SUBSET, SUPERSET or EQUAL, then the value of SHIFTRESULT is inverted (block **18F280**) using Table **18-5** illustrated in FIG. **18**Y. SHIFTRESULT is then returned (block **18F282**) to the FindComponents process illustrated in FIG. **18**E (block **18E118**).

If SHIFTRESULT is not equal to SUBSET, SUPERSET, or EQUAL (block **18F278**), CNSRESULT is set equal to SHIFTRESULT (block **18F279**). If CNS**2** is not true (block **18F122**), CNSRESULT is set equal to DISJOINT. Thereafter, the AlignComponents process recursively finds (block **18F284**) the components from EXPRESSION1 in EXPRESSION2 using the ShiftComponents process illustrated in FIG. **18**H and discussed in detail above. The ShiftComponents process return is SHIFTRESULT. Next, a test is made to determine if SHIFTRESULT is not DISJOINT (block **18F286**). If SHIFTRESULT is not DISJOINT, CNSRESULT is set equal to SHIFTRESULT (block **18F288**).

CNSRESULT is then returned (block **18F292**) to the FindComponents process illustrated in FIG. **18**E (block **18E118**). The result of the AlignComponents process (block **18E118**) is then returned (block **18E266**) to the FindComponents process shown in FIG. **18**D (block **18D094**).

Returning to FIG. **18**E, if **P2** is at the end of PTRLIST**2** (block **18E096**), which means that EXPRESSION1 is not at its end and/or EXPRESSION2 is at its end and has a trailing CONSUME group or is at its end and has no trailing CONSUME group, a test is made (block **18E310**) to determine if PTRLIST**2** ends with a CONSUME group. If the PTRLIST**2** array ends with a CONSUME group, RESULT is adjusted (block **18E312**) using Table **18-4** illustrated in FIG. **18**X. If the PTRLIST**2** array does not end with a CONSUME group, RESULT is set equal to DISJOINT (block **18E314**). If EXPRESSION1 and EXPRESSION2 are intersecting a FoundIntersection process (FIG. **18**M) illustrated and described below is executed (block **18E316**). The result is then returned (block **18E318**) to the FindComponentGroups process illustrated in FIG. **18**D (block **18D094**). The result returned from the FindComponents process (block **18D094**) is then returned (block **18D294**) as the result of the FindComponents process (FIG. **18**D) to the CompareComponentsGroups process shown in FIG. **18**A (block **18A072**).

Returning to FIG. **18**D, if **P1** is at the end of PTRLIST**1** (block **18D074**), EXPRESSION1 is either at its end with a trailing CONSUME group or is at its end without a trailing CONSUME group; and EXPRESSION2 is either at its end without a trailing CONSUME group, at the trailing CONSUME group, or not at its end and not at a trailing CONSUME group. The remainder of the CompareComponentsGroups process looks for these conditions, beginning with a test (block **18D296**) that determines if **P2** is at the last component in the PTRLIST**2** array. If **P2** is at the last component in the PTRLIST**2** array, **P2** is pointing at the trailing CONSUME group of EXPRESSION2. If PTRLIST**1** does not end with a trailing CONSUME group, the existing result must be merged with SUBSET. As a result, if **P2** is at the last component in the PTRLIST**2** array, a test (block **18D297**) is made to determine whether the PTRLIST**1** array ends with a CONSUME group. If the PTRLIST**1** array does not end with a CONSUME group, then the existing result must be adjusted. This is accomplished using Table **18-4** illustrated in FIG. **18**X (block **18D298**).

If **P2** is not at the last component in the PTRLIST**2** array (block **18D296**), EXPRESSION2 is either at its end without a trailing CONSUME group or not at its end and not at a trailing CONSUME group. It is known at this point that either EXPRESSION1 has a trailing "*" and EXPRESSION2 doesn't, or EXPRESSION1 has a trailing "*" and must CONSUME at least one more non-consume group from EXPRESSION2. If **P2** is not at the last component in the PTRLIST**2** array, a test (block **18D300**) is made to determine if the PTRLIST**1** array ends with a CONSUME group. If the PTRLIST**1** array does end with a CONSUME group, the existing result is adjusted (block **18D302**) using the Table **18-3** shown in FIG. **18**X. If the PTRLIST**1** array does not end with a CONSUME group, the existing result is set to DISJOINT (block **18D304**).

The value of the result obtained from the foregoing steps (blocks **18D302**, **18D304**, **18D298**, or **18D297**) is processed further if intersecting (block **18D306**) by executing the FoundIntersection process illustrated in FIG. **18**M and discussed below. The RESULT is then returned (block **18D308**) to the CompareComponentsGroups procedure call (block **18A072**) shown in FIG. **18**A.

The FoundIntersection process illustrated in flowchart form in FIG. **18**M begins with a test to determine if the

present result is DISJOINT (block **18M700**). If the present result is not DISJOINT, a possible intersection was not found and the FoundIntersection process ends. As a result, processing returns to the process that instituted the FoundIntersection process, i.e., the CompareComponentsGroups process (block **18D306**).

If the present result is not DISJOINT, a test (block **18M704**) is made to determine if the present result is SUBSET. If the present result is SUBSET, the state SWAPPED is tested (block **18M706**). If SWAPPED is true, the entire intersection process is aborted and the original EXPRESSION2 is returned (block **18M708**) as the intersection expression to the process that instituted the FoundIntersection process (i.e., block **18D306**, FIG. **18D**). If SWAPPED is not true, the entire intersection process is aborted and the original EXPRESSION1 is returned (block **18M710**) as the intersection expression to the process that instituted the FoundIntersection process.

If the present result is not SUBSET, a test (block **18M712**) is made to determine if the present result is EQUAL. If the expressions are equal, the intersection is either of the expressions, it doesn't matter which, because they are equal. If the present result is EQUAL (block **18M712**), the entire intersection process is aborted and the original EXPRESSION1 is returned (block **18M710**) as the intersection expression to the process that initiated the FoundIntersection process. If the present result is not EQUAL (block **18M712**), a test (block **18M714**) is made to determine if the present result is SUPERSET. If the present result is SUPERSET, a test (block **18M716**) is made to determine if SWAPPED is true. If SWAPPED is true, the entire intersection process is aborted and the original EXPRESSION1 is returned as the intersection expression (block **18M710**). If SWAPPED is not true, the entire intersection process is aborted and the original EXPRESSION2 is returned as the intersection expression (block **18M708**).

If the present result is not SUPERSET, the present result must be OVERLAP, which requires that a copy of the current intersection notation buffer be stored for later construction of an expression comprising part of the intersection. This process begins with a test (block **18M718**) to determine if there is an intersection notation buffer. If no intersection notation buffer exists, no consumption occurred. Since this buffer will not be modified, a global zero-length array can be used to be shared across all intersection operations to avoid memory waste. The zero-length intersection notation buffer is added (block **18M720**) to the intersection list and the FoundIntersection process ends (block **18M722**).

If an intersection notation buffer exists (block **18M718**), a test (block **18M724**) is made to determine if CURISECTPTR is greater than zero. If CURISECTPTR is greater than zero, a copy of the intersection notation buffer is added (block **18M726**) to the intersection list. The length of the copy is equal to the value of CURISECTPTR. If CURISECTPTR is not greater than zero (block **18M724**), a zero-length intersection notation buffer is added (block **18M720**) to the intersection list. Following either step (block **18M726** or block **18M720**) the FoundIntersection process ends and control is returned (block **18M722**) to the process that instituted the FoundIntersection process (i.e., block **18D306**, FIG. **18D**).

Returning to FIG. **18D**, after the FoundIntersection process is completed (block **18D308**), the final result is returned to the process that instituted the CompareComponentGroups process. i.e., the CompareExpressions process (block **18A072**, FIG. **18A**).

Returning to FIG. **18A**, if either of the expressions have extra data associated with them, they are compared (block **18A310**). The extra data comparison result of the comparison is then merged with the result returned by the CompareComponentsGroups process using Table **18-7** illustrated in FIG. **18Y**. The result returned by the CompareComponentsGroups process shown in the first column of Table **18-7** and the extra data comparison result is shown in the first row of Table **18-7**. For example, if the result returned by the CompareComponentsGroups process is SUPERSET (cell **18Y314**) and the extra data comparison result is SUBSET (cell **18Y316**), the merged result is DISJOINT (cell **18Y318**).

Intersecting Regular Expressions

The process for intersecting regular expressions is outlined by FIG. **18K**. This process is used to compare include/exclude set primitives, when the include/exclude set is comprised of regular expression primitives. A brief description of the figures for this process:

FIG. **18K** describes at the highest level the process of intersecting two expressions and initializes various variables used through the process.

FIG. **18L**, referenced by the comparison process described above, makes note of where a CONSUME group from one expression matched one or more components from the other expression.

FIG. **18M**, referenced by the comparison process described above, makes a copy of the notations created by the process described by FIG. **18L** for later use in constructing part of the final intersection expression.

FIG. **18N**, referenced at the end of the overall intersection process described by FIG. **18K**, describes at the highest level the process of constructing an intersection expression using the notations created by the process described by FIG. **18M**.

FIG. **18O** describes the process of constructing an intersection from a single sequence of notations.

FIG. **18P** describes the process of adjusting variables in the current state of the process to skip all of or part of the current component group.

FIGS. **18Q**–**18U** describe the process of merging component groups from the two expressions up to a point where consumption occurred. Two components are merged by selected the "more specific" of the two. For example, a MATCH component is selected over a CONSUME or SKIP component.

FIG. **18V** describes the process of copying consumed components from an expression, up to the point where consumption stopped.

Referring to FIG. **18K**, the IntersectExpressions process begins by normalizing EXPRESSION1 using the NormalizeExpressionInSeparateGroups process (block **18K350**) illustrated in FIG. **18B** and described in detail above. The array of component group descriptor codes returned by the NormalizeExpressionInSeparate-Component-Groups process is assigned the identifier PTRLIST**1**. Thus, PTRLIST**1** contains an array of the component group descriptor codes for EXPRESSION1. The array of individual characters for EXPRESSION1 is assigned the identifier EXPR1.

EXPRESSION2 is normalized and separated into component groups using the NormalizeExpressionInSeparateGroups process (block **18K352**). The array of component group descriptor codes returned by the NormalizeExpressionAndSeparateComponentGroups process is assigned to the identifier PTRLIST**2**. Thus, PTRL-

IST2 contains an array of component group descriptor codes for EXPRESSION2. The array of individual characters for EXPRESSION2 is assigned the identifier EXPR2.

After EXPRESSION1 and EXPRESSION2 have been normalized, a variable identifier called CURISECTPTR is initialized to zero (block **18K354**). CURISECTPTR is used to track the current location in an intersection notation buffer, reference being made to the NoteConsuming process (FIG. **18L**) and the prior description thereof for more details. After CURISECTPTR is initialized (block **18K354**), PTRLIST1 and PTRLIST2 are then compared using the CompareComponentGroups process (block **18K356**) with the value of CNS1 initialized to false, P1 initialized to 0, CNS2 initialized to false, P2 initialized to 0 and RESULT initialized to EQUAL. The CompareComponentGroups process is illustrated in FIG. **18D**, and described in detail above. If the expressions have extra data associated with them, the extra data is compared (block **18K358**). The result of the extra data comparison is merged with the expression comparison result returned from the CompareComponentGroups process (block **18K356**) using Table **18-7** shown in FIG. **18Y**. The operation of Table **18-7** is discussed above.

Next a test (block **18K360**) is made to determine if the merged comparison result (block **18K358**) is DISJOINT. If the merged comparison result is DISJOINT, the intersection is an empty expression. As a result, the intersection is returned (block **18K364**).

If the merged comparison result is not DISJOINT, an intersection is constructed using the ConstructIntersections process (block **18K366**) illustrated in flowchart form in FIG. **18N**.

Referring to FIG. **18N**, the ConstructIntersections process constructs a new expression for the intersection of the two original expressions by interpreting the notations made by the NoteConsuming process (FIG. **18L**) during the comparison discussed above. The ConstructIntersections process begins by initializing (block **18N368**) an empty expression for the intersection. The procedure next removes identical intersection notation buffers from the intersection list (block **18N368**). In this regard, it has been determined empirically that attempting to "memorize" comparison results and intersection notations from commonly reprocessed locations is not worthwhile, since it impedes the comparison process and the redundant notations are relatively inexpensive (in terms of memory) to keep anyway. As one skilled in the art will recognize, "memorize" is a computer science term. It means to save computation results to avoid performing the computation again, at the expense of having to look up results before performing a computation and saving (some of) the results afterwards. This approach typically is used for this type of algorithm, where computations are likely to repeat and are expensive in terms of computational resources. The ConstructIntersections procedure could be optimized by removing notations that have SUPERSET/SUBSET relationships. However, this is expensive relative to the cost of creating the redundant expressions in the final intersection. In an actual embodiment of this invention this has not been done because these cases are unlikely to occur.

After identical intersection notation buffers have been removed from the intersection list, a processing loop is entered. The processing loop begins with a test (block **18N372**) that determines if there are any more intersection notation buffers in the list. If the answer is a yes, the next intersection notation buffer is retrieved (block **18N374**) from the list. Then an intersection expression is constructed (block **18N376**) from the notations using the AddIntersection process illustrated in FIG. **18O**.

The AddIntersection process illustrated in FIG. **18O** begins by creating an empty expression (block **18O378**) and initializing variables designated **P1**, **N1**, **P2**, **I2**, and **N2** all to 0 (block **18O380**). EXPRESSION1 and EXPRESSION2 are advanced (blocks **18O382** and **18O383**) by positive infinity using the Advance process illustrated in FIG. **18P**.

The advance process (FIG. **18P**) uses a variable N to refer to the number of components to advance as specified by the referring process, i.e., the process calling the Advance process. The variables used within the Advance process are of the form Ix, which refers to either **I1** or **I2**, depending on the referring flowchart. This could alternatively be expressed by first swapping the orientation, performing the Advance process, and then reswapping the orientation. For brevity, these steps have been omitted in favor of the foregoing.

The first step in the Advance process is a test (block **18N384**) to determine if N is greater or equal to Nx (i.e., **N1** or **N2** as the case may be). This test checks to see if the length of the current group has been exhausted. The value of N or Nx may be positive infinity. For the purposes of the Advance process, if both N and Nx are positive infinity, N and Nx are considered equal. In one implementation of the invention, positive infinity is replaced with a value that is larger than the longest possible expression character length, so that advancing by this amount forces this flowchart to skip to the next component group. In an actual embodiment of the invention, this value is the Java constant java/lang/Integer.MAX__VALUE.

If N is not greater than or equal to Nx (block **18N384**), then the Advance process advances within a group by adding N to Ix and subtracting N from Nx (block **18N386**). The processing then returns (block **18N388**) to the process calling or instituting the Advance process (i.e., block **18O382** in FIG. **18O**).

If N is greater than or equal to Nx (block **18N384**), the Advance process determines if all of the groups in the expression have been exhausted. First a test (block **18N390**) is made to determine if Px is equal to the length of the PTRLISTx. If the answer is yes, a variable designated TYPEx is set to the value END (block **18N392**) and processing returns to the process calling or instituting the Advance process.

If Px is not equal to the length of the PTRLISTx, the Advance process advances to the next group by setting TYPEx to the type of group at Px (block **18N396**). Next a test is made to determine if TYPEx is CONSUME. If TYPEx is CONSUME, N is set equal to positive infinity (block **18N400**). The value of Px is then advanced to the start of the next component group (block **18N402**), the Advance process ends (block **18N404**) and processing returns to the process that called or initiated the Advance process.

If TYPEx is not CONSUME (block **18N398**), Nx is set equal to the length of the component group (block **18N406**). Then a test (block **18N408**) is made to determine if TYPEx is SKIP. If TYPEx is SKIP, Ix is set equal to 0 (block **18N410**). If TYPEx is not SKIP, Ix is set equal to the value of the character index of the start of the character group (block **18N412**). Following the setting of Ix (block **18N410** or block **18N412**), Px is advanced to the start of the next component group (block **18N402**), the Advance process ends and processing returns to the process that called or initiated the Advance process, e.g., the AddIntersection process (block **18O382**, FIG. **18O**).

Returning to the AddIntersection process (FIG. **18O**), after EXPRESSION1 and EXPRESSION2 have been

advanced by positive infinity using the Advance process, a test (block **18O383**) is made to determine if there are any more notations to process. If there are no more notations to process, P1STOP, I1STOP, P2STOP, I2STOP are set equal to positive infinity (block **18O417**) and the remaining groups are merged (block **180419**) using the IntersectGroups process (FIG. **18**Q). The expression is then returned (block **18O418**) to the process initiating or calling the AddIntersection process.

If there are more notations to process (block **18O416**), the next notation is retrieved (block **18O420**) along with CNSSTART, which is a variable representing the group index following the group that did the consuming, PSTOP, which is a variable representing the group index following the group that was consumed from, and ISTOP, which is a variable representing the index into the group where consumption stopped from the notation. Next a test (block **18O422**) is made to determine if CNSSTART is greater than the length of PTRLIST**1**. If CNSSTART is greater than the length of PTRLIST**1**, SWAPPED is set to true (block **18O424**). The length of PTRLIST**1** is then subtracted from CNSSTART (block **18O426**), then a variable designated P**2**STOP is set equal to CNSSTART and a variable designated I**2**STOP is set equal to 0 (block **18O428**). Thereafter the value of a variable designated P**1**STOP is set equal to PSTOP and the value of a variably designated I**1** STOP is set equal to ISTOP (block **18O430**).

If CNSSTART is not greater than the length of the PTRLIST**1** (block **18O422**), SWAPPED is set to false (block **18O432**). Then P**1**STOP is set equal to CNSSTART and I**1**STOP is set equal to 0 (block **18O434**). Next, P**2**STOP is set equal to PSTOP and I**2**STOP is set equal to ISTOP (block **18O436**). After the final settings have occurred (block **18O430** or block **18O436**), the groups are merged up to the point of consumption using the IntersectGroups process illustrated in flowchart form in FIG. **18**Q (block **18O438**). After the IntersectGroups process is completed, a test is made (block **18O900**) to determine if SWAPPED is true. If SWAPPED is true, the consumed groups from EXPRESSION1 are copied **(18O492)** using the CopyGroups process (FIG. **18**V). If SWAPPED is false, the consumed groups from EXPRESSION2 are copied **(18O492)** using the CopyGroups process (FIG. **18**V). After the consumed groups are copied (block **18O492** or **18O494**), the process cycles to the any more notations test **(18O416)**.

The IntersectGroups process (FIG. **18**Q) begins by checking EXPRESSION1 for a stopping point using the CheckIfShouldStopIntersecting process illustrated in flowchart form in FIG. **18**R (block **18Q440**). The CheckIfShouldStopIntersecting process (FIG. **18**R) uses variables in the form of Ix, meaning either I**1** or I**2**. The variable used is actually determined by the referring process. This could also be expressed by first swapping the orientation of the expression, performing the CheckIfShouldStopIntersecting process, and then reswapping the orientation. For brevity, these alternative steps have not been included. The notations used in the CheckIfShouldStopIntersecting process indicate complete consumption of a group with I+N, where I is set to 0 for SKIP groups or for character groups is the starting character index of the group. In the CheckIfShouldStopIntersecting process described here, the process will already have advanced to the next group when the group's length is exhausted.

The CheckIfShouldStopIntersecting process begins with a test (block **18R442**) that determines if Px is greater than PxSTOP. If Px is greater than PxSTOP, a stop message is returned (block **18R442**) to the process that called or initiated the CheckIfShouldStopIntersecting process.

If Px is not greater then PxSTOP (block **18R442**), a test (block **18R446**) is made to determine if Px is equal to PxSTOP. If Px is equal to PxSTOP, a variable designated Nx is set equal to the value of IxSTOP minus the value of Ix (block **18R442**). Next a test (block **18R450**) is made to determine if Nx is greater than 0. If Nx is not greater than 0, the stop message is returned (block **18R442**) to the process that called the CheckIfShouldStopIntersecting process. If Nx is greater than **0**, or if Px is not equal to PxSTOP, a continue message is returned (block **18R452**) to the process that called or initiated the CheckIfShouldStopIntersecting process.

Returning to FIG. **18**Q, after EXPRESSION1 has been checked for a stopping point using the CheckIfShouldStopIntersecting process (block **18Q440**), a test (block **18Q454**) is made to determine if a stopping point has been reached. If a continue message was received from the CheckIfShouldStopIntersecting process, EXPRESSION2 is checked (block **18Q456**) for a stopping point using the CheckIfShouldStopIntersecting process illustrated in FIG. **18**R and described above. Thereafter a test (block **18Q458**) is made to determine if the message returned from the CheckIfShouldStopIntersecting process is stop or continue. If the returned message was continue, the group types are merged (block **18Q460**) using Table **18-6**, illustrated in FIG. **18**Y.

After the group types have been merged (block **18Q460**), a test (**18Q470**) is made to determine if the result of the merger is stop merging. If the result of the merger is stop merging (block **18Q470**) or if stopping points are reached when EXPRESSION1 or EXPRESSION2 were checked for stopping points (block **18Q454** or block **18Q458**), the IntersectGroups process ends and processing returns to the process that called or initialized the IntersectGroups process. If the merger return is not Stop merging (block **18Q470**), then the operation listed in Table **18-6** resulting from the merger is performed (block **18Q474**).

Three of the processes listed in Table **18-6** are illustrated in FIG. **18**S. These are CopyMatch**1**, CopyMatch**2** and CopySkips. CopyMatch**1** (block **18S476**) begins by setting a variable designated NADVANCE (block **18S478**) to the minimum of N**1** and N**2**. Thereafter the NADVANCE characters are copied to the intersection expression from EXPRESSION1 starting at I**1** (block **18S480**). CopyMatch**2** (block **18S482**) begins by setting NADVANCE to the minimum of N**1** and N**2**. Thereafter the NADVANCE characters are copied to the intersection expression from EXPRESSION2 starting at I**2** (block **18S486**). CopySkips (block **18S488**) begins by setting NADVANCE to the minimum of N**1** and N**2** (block **18S490**). Then a number of "?" characters equal to NADVANCE are added to the intersection expression (block **18S492**).

Following the appropriate one of the foregoing operations (blocks **18S480**, **18S486** or **18S492**), EXPRESSION1 is advanced (block **18S494**) by NADVANCE using the Advance process illustrated in FIG. **18**P and discussed above. EXPRESSION2 is then advanced (block **18S496**) by NADVANCE using the Advance process illustrated in FIG. **18**P and discussed above. The related process then ends and processing returns (block **18S498**) to the calling or initiating process, i.e., the IntersectGroups process shown in FIG. **18**Q and described above.

Three of the processes listed in Table **18-6** are illustrated in FIG. **18**T. They are TrailConsume**1**, TrailConsume**2**, and CopyConsume. TrailConsume**2** (block **18T500**) begins with a test (block **18T504**) to determine if P**2** is at the end of

PTRLIST2. If P2 is at the end of PTRLIST2, a test (block 18T506) is made to determine if PTRLIST1 ends with a CONSUME group. If PTRLIST1 does not end with a CONSUME group, then EXPRESSION1 is advanced (block 18T508) by positive infinity using the Advance process illustrated in FIG. 18P and described above. If P2 is not at the end of PTRLIST2 (block 18T504) or if PTRLIST1 does not end with a consume group (block 18T506), a "*" is appended (block 18T510) to the intersection expression and EXPRESSION1 is advanced by positive infinity using the Advance process (block 18T508).

TrailConsume1 (block 18T502) begins with a test (block 18T512) to determine whether P1 is at the end of PTRLIST1. At this point, if there is trailing consumer, a trailing "*" is not coded if EXPRESSION2 does not have a trailing "*". This case happens because trailing consumption is not indicated by the notations from the CompareComponents process (FIG. 18D) and the FindComponents process (FIG. 18E) because these processes stop at trailing consumers. (A similar case exists for EXPRESSION1 in the TrailConsume2 process described above).

If P1 is at the end of PTRLIST1 (block 18T512), a test (block 18T514) is made to determine if PTRLIST2 ends with a consume group. If PTRLIST2 does not end with a consume group, EXPRESSION1 is advanced by positive infinity using the Advance process (block 18T508). If P1 is not at the end of PTRLIST1 (block 18T512), or if PTRLIST2 does end with a consume group (block 18T514) a "*" is appended to the intersection expression (block 18T510) and EXPRESSION1 is advanced by infinity using the Advance process (block 18T508).

Copy Consume (block 18T516) begins by setting NADVANCE equal to positive infinity (block 18T518). A "*" is then added to the intersection expression (block 18T520) and EXPRESSION1 is advanced by positive infinity using the Advance process (block 18T508).

After EXPRESSION1 is advanced to positive infinity using the Advance process (block 18T508), EXPRESSION2 is advanced by positive infinity using the Advance process illustrated in FIG. 18P and described above. After EXPRESSION1 and EXPRESSION2 have been advanced to positive infinity processing returns to the process that initiated or called to selected one of the process shown in FIG. 18T, i.e., the IntersectGroups process (FIG. 18Q).

FIG. 18U illustrates the Advance1 (block 18U526) and the Advance2 (block 18U528) processes listed in Table 18-6. Advance1 begins by advancing (block 18U530) EXPRESSION1 by positive infinity using the Advance process (FIG. 18P), which is described above. At this point, if the comparison is "*" versus "?", or "*" versus a character, a "*" is inserted if the "?" group is preceded by a "* " or if the character group is at the beginning and is preceded by a "*". This is done first conducting a test (block 18U532) to determine if the component group before the group preceding the group in the PTRLIST2 at P2 is a consume group. If this group is not a consume group, the Advance1 process ends and processing returns to the process that called or initiated the Advance1 process, i.e., the IntersectGroups process. If this group is a consume group (block 18U532), a test (block 18U536) is made to determine if the component group preceding the group in PTRLIST2 at P2 is a skip group. If this group is a skip group, a "*" is appended (block 18U538) to the intersection expression and processing returns to the process that called or initiated the Advance1 process. If this group is not a skip group (block 18U536), a test (block 18U540) is made to determine if I2 is at the start of the character group preceding the group in PTRLIST2 at P2. If the answer is yes, a "*" is added to the intersection expression and processing returns to the process that called the Advance1 process. If the answer is no, processing returns to the process that called the Advance1 process.

The Advance2 process (block 18U528) is very similar to the Advance1 process (block 18U526) and done for similar reasons. First, EXPRESSION2 is advanced (block 18U542) by positive infinity using the Advance process (FIG. 18P). Next, a test (block 18U544) is made to determine if the component group before the group preceding the group in PTRLIST1 at P1 is a consume group. If the answer is no, processing returns (block 18U534) to the process that initiated or called the Advance2 process, i.e., the IntersectGroups process (FIG. 18Q). If the answer is yes, a test (block 18U546) is made to determine if the component group preceding the group in PTRLIST1 at P1 is a skip group. If the answer is yes, a "*" is appended to the intersection expression (block 18U538) and processing returns to the process that called the Advance2 process (block 18U534). If the answer is no, a test (block 18U548) is made to determine if I1 is at the start of the character group preceding the group in PTRLIST1 at P1. If the answer is yes, a "*" is appended to the intersection expression (block 18U538) and processing returns to the process that called the Advance2 process (block 18U534). If the answer is no, processing returns to the process that called the Advance2 process (block 18U534).

The CopyGroups process is illustrated in FIG. 18V. The variables used in FIG. 18V are in the form of "Ix", meaning either I1 or I2. The variable to use is determined by the expression being processed. This could also be expressed by first swapping the orientation, performing the CopyGroups process, and then reswapping the orientation. The first step in the CopyGroups process is a test (block 18V550) to determine if the variable Px is less than the variable PSTOP. If Px is less than PSTOP, a test (block 18V552) is made to determine if TYPEx is CONSUME. If TYPEx is CONSUME, a "*" is appended (block 18V554) to the intersection expression and EXPRESSIONx is advanced (block 18V556) using the Advance process illustrated in FIG. 18P and discussed above. If TYPEx is not CONSUME (block 18V552), then a test (block 18V558) is made to determine if TYPEx is SKIP. If TYPEx is not SKIP, then Nx characters are copied from EXPRESSIONx (block 18V560) starting at the position Ix. Thereafter, EXPRESSIONx is advanced by the value of Nx using the Advance process shown in FIG. 18P. If TYPEx is SKIP, then Nx "?" characters are appended (block 18V562) to the intersection expression and then EXPRESSIONx is advanced (block 18V556) by the value of the variable Nx using the Advance process shown in FIG. 18P. After the EXPRESSIONx is Advanced, the process repeats, starting at the Px less than PSTOP test (block 18V550).

If Px is not less than PSTOP (block 18V550), a test (block 18V558) is made to determine if Px is equal to PSTOP. (This portion of the CopyGroups process copies the starting portion of the partially consumed group.) If Px is equal to PSTOP, the variable N is set equal to the value of ISTOP minus Ix (block 18V560). N is then tested to see if it is greater than 0 (block 18V562).

If Px is not equal to PSTOP (block 18V558) or N is not greater than 0 (block 18V562), the CopyGroups process ends and processing returns to the process that called the CopyGroups process.

If N is greater than 0 (block 18V562), a test is performed to determine if TYPEx is SKIP (block 18V564). If TYPEx

is SKIP, N "?" characters are appended (block **18V566**) to the intersection expression. If TYPEx is not SKIP, N characters are copied from EXPRESSIONx starting from Ix (block **18V568**). Thereafter EXPRESSIONx is advanced (block **18V570**) by the value of N using the Advance process shown in FIG. **18P** and processing returns (block **18V572**) to the process that called the CopyGroups process.

Compare Two Lists of Wildcards

The process for comparing wildcards included in the sets of permissions being evaluated, i.e., the requested permission set and the user permission set is illustrated in functional flow form in FIG. **18W**. Note that this process must be used instead of the process described by FIG. **14D**, since two wildcards may have overlapping values. First, each expression in one set of permissions, e.g., LIST**1**, is compared (block **18W574**) against each expression in the other set of permissions, e.g., LIST**2**, using the CompareComponents-Groups process illustrated in FIG. **18D** and described above. The results in each row are then incrementally merged (block **18W576**) using Table **18-1** (FIG. **18X**). The results of the merge operation are then stored in a vector (block **18W576**). Next, a variable designated ROWSRESULT is assigned (block **18W578**) to the result of incrementally merging the results in the vector using Table **18-2** (FIG. **18X**). Next, the results in each column are then incrementally merged (block **18W584**) using Table **18-2** (FIG. **18X**). The result of this incremental merge are stored in a vector (block **18W584**). Next, a variable designated COLSRESULT is assigned (block **18W586**) to the result of incrementally merging the results in a vector using Table **18-1** FIG. **18X**). Then, a test is made (block **18W588**) to determine if ROWSRESULT is equal to COLSRESULT. If ROWSRESULT equals COLSRESULT, a variable designated RESULT is set equal to ROWSRESULT (block **18W390**). If ROWSRESULT does not equal COLSRESULT, a test (block **18W592**) is made to determine if ROWSRESULT is OVERLAP. If ROWSRESULT is OVERLAP, RESULT is set equal to COLSRESULT (block **18W594**). If ROWSRESULT is not OVERLAP, a test (block **18W596**) is made to determine if COLSRESULT is OVERLAP. If the COLSRESULT is OVERLAP, RESULT is set equal to ROWSRESULT (block **18W578**). If COLSRESULT is not OVERLAP, RESULT is set equal to EQUAL (block **18W600**).

Example comparisons and intersections according to the method illustrated in FIGS. **18A–18Y** are shown in Table **18-8** FIGS. **18Z–AA**). DISJOINT expressions do not have intersections. Otherwise, where an intersection has been omitted, the comparison result is either EQUAL, SUPERSET, or SUBSET, in which case the intersection is the "lesser" of the two expressions. Semi-colons delimit expression lists.

Comparison of Include/Exclude Set Primitives

The preferred method employed by the invention for comparing primitives of the include/exclude pair type is illustrated in functional flow diagram form in FIGS. **19A–19I**. Include/exclude pairs are discussed above with reference to FIG. **9H**. Briefly, an include/exclude pair consists of an include expression and a paired exclude expression. The include expression defines a set of things from which a subset of things defined by the exclude expression is taken away. For instance, if an include expression defined a set of things a, b, c, and the exclude expression defined a subset of things b, then the include/exclude pair results in a subset a, c. In the Include/Exclude Sets primitive process shown in FIGS. **19A–19I**, the variables **I1** and **E1** represent the include/exclude pair from a first permission set, and the variables **I2** and **E2** represent the include/exclude pair from a second permission set. While the method illustrated in FIGS. **19A–19I** can be used to compare any two sets, in the context of the present invention, the **I1/E1** include/exclude pair can be thought of as being part of the requested permission set, and the **I2/E2** include/exclude pair can be thought of as part of the user permission set.

FIG. **19A** begins by comparing (block **19 A010**) **I1** against **I2**. The expression **I1** and **I2** can be any of the primitive types discussed above. The comparison of **I1** against **I2** is specific to the type of primitive and uses the method described above for each primitive type. For instance, regular expressions are commonly used in include/exclude pairs to define sets of files. The include expression could be something such as "*.txt" and the exclude expression could be "mydoc.txt" which would define a set of files that includes all files with the last four characters ".txt" except the file "mydoc.txt". The second set could include a corresponding include/exclude pair that defines the include as "*.txt" and the exclude expression as "mydoc*.txt". In this example, while the include expressions are equal, it is apparent that the exclude expression of the second set defines many more documents than the exclude expression of the first set. The method and system of comparison illustrated in FIGS. **19A–I** provide a directional comparison result in situations such as this.

A variable designated **I1**CMP**I2** is set equal to the result of the comparison of **I1** against **I2** using the comparison method discussed above for the constituent primitives (block **19A010**). Thereafter, a test (block **19A012**) is made to determine if **I1**CMP**I2** is EMPTY, EMPTY SUBSET, EMPTY SUPERSET, or DISJOINT. If the answer is yes, **I1**CMP**I2**, i.e., EMPTY, EMPTY SUBSET, EMPTY SUPERSET, or DISJOINT is returned (block **19A014**) to the process calling the Include/Exclude Sets primitive process. If **I1**CMP**I2** is not EMPTY, EMPTY SUBSET, EMPTY SUPERSET, or DISJOINT, **E1** is compared against **E2** and the variable designated **E1**CMP**E2** is set equal to the result of the comparison (block **19A016**). Then a test (block **19A018**) is made to determine if **E1**CMP**E2** is EMPTY. If **E1**CMP**E2** is EMPTY, **I1**CMP**I2** is returned (block **19A020**) to the process calling Include/Exclude Sets primitive process calling. If **E1**CMP**E2** is not EMPTY (block **19A018**), then a test (block **19A022**) is made to determine if **I1**CMP**I2** is OVERLAP. If **I1**CMP**I2** is OVERLAP, the IncludesOverlap process illustrated in functional flow form in FIG. **19B** is performed (block **19A024**).

The IncludesOverlap process begins with a test (block **19B026**) to determine if **E1**CMP**E2** is EQUAL. If **E1**CMP**E2** is EQUAL, a variable designated XI**1**I**2** is set equal to the intersection of **I1** and **I2** (block **19B028**). Computing the intersection of the expression is specific to the constituent primitive. For instance, if **I1** and **I2** are regular expressions, the intersection is computed in the manner illustrated in FIG. **18K** and related figures described above. Next, **E1** is compared against XI**1**I**2** (block **19B030**). Then a test (block **19B032**) is made to determine if the result of the comparison is EQUAL. If the result of the comparison is EQUAL, DISJOINT is returned (block **19B034**) to the process calling the IncludesOverlap process. If the result of the comparison is not EQUAL, OVERLAP is returned (block **19B036**) to the calling process.

If **E1**CMP**E2** is not EQUAL (block **19B026**), a test (block **19B038**) is made to determine if **E1**CMP**E2** is SUBSET. If

E1CMPE2 is SUBSET, a SetOpD process illustrated in functional flow diagram form in FIG. 19F is performed (block 19B040).

The first step in the SetOpD process illustrated in FIG. 19F is to set XI1I2 to the intersection of I1 and I2 (block 19F042). Then E2 is compared against XI1I2 (block 19F044). Then a test (block 19F046) is made to determine if the result of the comparison is EQUAL or SUPERSET. If the result of the comparison is EQUAL or SUPERSET, DISJOINT is returned (block 19F048) to the process calling the SetOpD process. If the result of the comparison is not EQUAL or SUPERSET, a variable designated UE2XI1I2 is set equal to the union of E2 and XI1I2 (block 19F050). The union represents an aggregation of the set of things contained in the E2 expression and the set of things contained in the XI1I2 expression. Next I2 is compared against UE2I1I2 (block 19F052). Then a test (block 19F054) is made to determine whether the result of the comparison is EQUAL. If the result of the comparison is EQUAL, SUPERSET is returned (block 19F056) to the calling process. If the result of the comparison is not EQUAL, OVERLAP is returned (block 19F058) to the calling process.

Returning to FIG. 19B, if E1CMPE2 is not SUBSET (block 19B038), a test (block 19B060) is made to determine if E1CMPE2 is OVERLAP. If E1CMPE2 is OVERLAP, a SetOpE process illustrated in functional flow diagram from FIG. 19G is performed (block 19B062).

The first step of the SetOpE process illustrated in FIG. 19G is to set a variable designated UE1E2 equal to the union of E1 and E2 (block 19G300). The next step is to set XI1I2 equal to the intersection of I1 and I2 (block 19G302). Then UE1E2 is compared against the value of the variable XI1I2 (block 19G304). If the result of the comparison is EQUAL or SUPERSET (block 19G306), DISJOINT is returned (block 19G308) to the process calling the SetOpE process. If the result of the comparison is not EQUAL or SUPERSET, a variable designated UE1XI1I2 is set equal to the union of E1 and XI1I2, UE1XI1I2 is compared to I1; and a variable designated E1COMPLETE is set to true if the result of the comparison is EQUAL (block 19G310). Next, a variable designated UE2XI1I2 is set equal to the union of E2 and XI1I2; UE2XI1I2 is compared to I2; and a variable designated E2 COMPLETE is set to true if the result of the comparison is EQUAL (block 19G312).

Next, a test (block 19G314) is made to determine if both E1COMPLETE and E2COMPLETE are false. If E1COMPLETE and E2COMPLETE are both false, OVERLAP is returned to the calling process (block 19G316). If E1COMPLETE and E2COMPLETE are not both false, a variable designated XE1XI1I2 is set equal to the intersection of E1 and XI1I2 (block 19G318). Then, a variable designated XE2XI1I2 is set equal to the intersection of E2 and XI1I2 (block 320). Thereafter, a variable designated CMP is set equal to the result of comparing XE1XI1I2 and XE2XI1I2 (block 19G322). Next, a test (block 19G324) is made to determine if both E1COMPLETE and E2COMPLETE are true, if both E1COMPLETE and E2COMPLETE are true, a test (block 19G326) is made to determine if CMP is EQUAL. If CMP is EQUAL, EQUAL is returned to the calling process (block 19G328). If CMP is not EQUAL, a test (block 19F330) is made to determine if CMP is SUBSET. If CMP is SUBSET, SUPERSET is returned to the calling process (block 19G322). If CMP is not SUBSET, a test is made to determine if CMP is SUPERSET (block 19G334). If CMP is SUPERSET, SUBSET is returned to the calling process (block 19G336). If CMP is not SUPERSET, OVERLAP is returned to the calling process (block 19G338).

If both E1COMPLETE and E2COMPLETE are not true (block 19G324), a test (block 19G340) is made to determine if E1COMPLETE is true. If E1COMPLETE is true, a test (block 19G342) is made to determine if CMP is EQUAL or SUPERSET. If CMP is EQUAL or SUPERSET, SUBSET is returned to the calling process (block 19G344). If CMP is not EQUAL or SUPERSET, OVERLAP is returned to the calling process (block 19G338). If E1COMPLETE is false (block 19G340), a test (block 19G346) is made to determine if E2COMPLETE is true. If E2COMPLETE is true, a test (block 19G348) is made to determine if CMP is EQUAL or SUBSET. If CMP is EQUAL or SUBSET, SUPERSET is returned to the calling process (block 19G350). If either E2COMPLETE is false (block 19G346) or CMP is not EQUAL or SUBSET (block 19G348), OVERLAP is returned to the calling process (block 19G338).

Returning to FIG. 19B, if E1CMPE2 is not OVERLAP (block 19B060), a test (block 19B074) is made to determine if E1CMPE2 is DISJOINT, EMPTY SUPERSET, or EMPTY SUBSET. If E1CMPE2 is DISJOINT, EMPTY SUPERSET or EMPTY SUBSET, a SetOpF process illustrated in functional flow diagram form in FIG. 19H is performed (block 19B076).

The first step of the SetOpF process is illustrated in FIG. 19H is to set UE1E2 equal to the union of E1 and E2 (block 19H078). Then, XI1I2 is set equal to the intersection of I1 and I2 (block 19H080). Next, UE1E2 is compared to XI1I2 (block 19H082). Then a test (block 19H084) is made to determine if the result of the comparison is EQUAL or DISJOINT. If the result of the comparison is EQUAL or DISJOINT, DISJOINT is returned to the calling process (block 19H086). If the result of the comparison is not EQUAL or DISJOINT (block 19H084) E1 is compared to XI1I2 and a variable designated E1OUTSIDE is set to true if the result of the comparison is DISJOINT or EMPTY SUBSET (block 19H088). Then, E2 is compared to XI1I2 and a variable designated E2OUTSIDE is set true is the result of the comparison is DISJOINT or EMPTY SUBSET. Next, a test (block 19H092) is made to determine if both E1OUTSIDE and E2OUTSIDE are false. If both E1OUTSIDE and E2OUTSIDE are false, OVERLAP is returned to the calling process (block 19H094).

If both E1OUTSIDE and E2OUTSIDE are not false, UE1X1I1I2 is set to the union of E1 and XI1I2; UE1XI1I2 is compared against I1; and E1COMPLETE is set to true if the result of the comparison is EQUAL (block 19H096). Next UE2X1I1I2 is set to the union of E2 and XI1I2; UE2XI1I2 is compared against I2; and E2COMPLETE is set true if the result of the comparison is EQUAL (block 19H098). Then E1OUTSIDE (block 19H088), E2OUTSIDE (block 19H090) E1COMPLETE (block 19H096) and E2COMPLETE (block 19H098), are used to determine the result of the SetOpF process from Table 19-3 (FIG. 19I). For instance, if E1OUTSIDE is true (cell 19I102), E1COMPLETE is false (cell 19I104), E2OUTSIDE is true (cell 19I106) and E2COMPLETE is true (cell 19I108), the result is SUPERSET (cell 19I110). Returning to FIG. 19H, the value obtained from Table 19-3 is returned to the calling process (block 19H112).

Returning to FIG. 19B, if E1CMPE2 is not DISJOINT, EMPTY SUPERSET, or EMPTY SUBSET (block 19B074), I1 is swapped with I2, E1 is swapped with E2 and E1CMPE2 is inverted using Table 19-2 (FIG. 19I) (block 19B116). Next, the SetOpD process illustrated in FIG. 19F and described above is performed (block 19B118). The result returned by the SetOpD process is inverted (block 19B120) using Table 19-2 FIG. 19I). The inverted result is returned to the calling process (block 19B122).

Returning to FIG. **19A** if I1CMPI2 is not OVERLAP (block **19A022**), a variable designated OP is set to a value obtained from Table **19-1** FIG. **19I** using I1CMPI2 for the row value and E1CMPE2 for the column value. For instance, if I1CMPI2 is SUPERSET (cell **19I128**) and E1CMPE2 is DISJOINT (cell **19I130**), OP is set to Operation C (cell **19I132**).

Next a test (block **19A134**) is made to determine if OP is SUBSET, EQUAL, SUPERSET, or OVERLAP. If OP is SUBSET, EQUAL, SUPERSET or OVERLAP, OP is returned to the calling process (block **19A136**). If OP is not SUBSET, EQUAL, SUPERSET or OVERLAP, a test (block **19A138**) is made to determine if OP is operation A. If OP is Operation A, the SetOpA process illustrated in functional flow form in FIG. **19C** is performed (block **19A140**).

The first step in the SetOpA process (FIG. **19C**) is to set a variable designated E2CMPI1 to the result of comparing E2 against I1 (block **19C142**). Then a test (block **19C144**) is made to determine if E2CMPI1 is EQUAL or SUPERSET. If E2CMPI1 is EQUAL or SUPERSET, DISJOINT is returned to the calling process (block **19C146**). If E2CMPI1 is not EQUAL or SUPERSET, a test (block **19C148**) is made to determine if E2CMPI1 is SUBSET.

If E2CMPI1 is SUBSET, UE1E2 is set to the union of E1 and E2 (block **19C150**). Then UE1E2 is compared against I1 (block **19C152**). If the result of the comparison is EQUAL or SUPERSET (block **19C154**), DISJOINT is returned to the calling process (block **19C156**). If the result of the comparison is not EQUAL or SUPERSET, OVERLAP is returned to the calling process (block **19C158**).

If E2CMPI1 is not SUBSET (block **19C148**), a test (block **19C160**) is made to determine if E2CMPI1 is DISJOINT. If E2CMPI1 is DISJOINT, a test (block **19C162**) is made to determine if E1CMPE2 is EMPTY SUBSET. If E1CMPE2 is not EMPTY SUBSET, SUBSET is returned to the calling process (block **19C164**).

If E1CMPE2 is EMPTY SUBSET, UI1E1 is set to the union of I1 and E2 (block **19C166**). Then UI1E2 is compared against the value of I2 (block **19C168**). Thereafter a test (block **19C170**) is made to determine if the result of the comparison is EQUAL. If the result of the comparison is EQUAL, EQUAL is returned to the calling process (block **19C172**). If the result of the comparison is not EQUAL, SUBSET is returned to the calling process (block **19C174**).

If E2CMPI1 is not equal to DISJOINT (block **19C160**), the SetOpA-overlap process illustrated in flow diagram form in FIG. **19D** is performed (block **19C176**). The first step in the SetOpA-Overlap process (block **19D178**) is to determine if E1CMPE2 equals SUBSET. If E1CMPE2 is SUBSET, UE2I1 is set to the union of E2 and I1; UE2I1 is compared against I2; and a variable designated I2MI1EXCLUDED is true if the result of the comparison is EQUAL (block **19D180**). Next, XE1I1 is set to the intersection of E2 and I1; XE2I1 is compared against E1; and a variable designated E1E2EQINISECT is set true if the result of the comparison is EQUAL (block **19D182**). Then a test (block **19D184**) is made to determine if I2MI1EXCLUDED is true. If I2MI1EXCLUDED is true, and a test (block **19D186**) is made to determine if E1E2EQINISECT is true. If E1E2EQINISECT is true, EQUAL is returned to the calling process (block **19D188**). If E1E2EQINISECT is not true, SUPERSET is returned to the calling process (block **19D190**).

If I2MI1EXCLUDED is not true (block **19D184**), a test (block **19D192**) is made to determine if E1E2EQINISECT is true. If E1E2EQINISECT is true, SUBSET is returned to the calling process (block **19D194**). If E1E2EQINISECT is not true, OVERLAP is returned to the calling process (block **19D196**).

If E1CMPE2 is not SUBSET (block **19D178**), a test (block **19D198**) is made to determine if E1CMPE2 is EMPTY SUBSET. If E1CMPE2 is EMPTY SUBSET, UI1E2 is set to the union of I1 and E2 and UI1E2 is compared against I2 (block **19D200**). Then a test (block **19D202**) is made to determine if the result of the comparison is EQUAL. If the result of the comparison is EQUAL, SUPERSET is returned to the calling process (block **19D204**). If the result of the comparison is not EQUAL, OVERLAP is returned to the calling process (block **19D206**).

If E1CMPE2 is not EMPTY SUBSET (block **19D198**), UE1E2 is set to the union of E1 and E2, and UE1E2 is compared against I1 (block **19D208**). At this point, I1 is a SUBSET of I2, so that the intersection is I1. Next, a test (block **19D210**) is made to determine if the result of the comparison is EQUAL or SUPERSET. If the result of the comparison is not EQUAL or SUPERSET, DISJOINT is returned to the calling process (block **19D212**). If the result of the comparison is EQUAL or SUPERSET, a test (block **19D214**) is made to determine if E1CMPE2 is OVERLAP. If E1CMPE2 is not OVERLAP, OVERLAP is returned to the calling process (block **19D216**). If E1CMPE2 is OVERLAP, XE2I1 is set to the intersection of E2 and I1, and XE2I1 is compared against E1 (block **19D218**). Then a test (block **19D220**) is made to determine if the result of the comparison is SUBSET. If the result of the comparison is not SUBSET, OVERLAP is returned to the calling process (block **19D222**). If the result of the comparison is SUBSET, SUBSET is returned to the calling process (block **19D224**).

Referring to FIG. **19A**, if OP is Operation C (block **19D226**), I1 is swapped with I2 and E1 is swapped with E2 (block **19A228**); and E1CMPE2 is inverted using Table **19-2** illustrated in FIG. **19I**. Then the SetOpA process illustrated in FIG. **19C** and discussed above is performed using the swapped and inverted values (block **19A230**). The result returned from the SetOpA process is then inverted (block **19A232**) using Table **19-2** shown in FIG. **19I** and the result is returned to the calling process (block **19A234**). If OP is not Operation C (block **19A226**), the SetOpB process illustrated in flow diagram form in FIG. **19E** is performed (block **19A236**).

The first step in the SetOpB process (FIG. **19E**) is to set UE1E2 to the union of E1 and E2; then UE1E2 is compared (block **19E066**) against either I1 or I2 (I1 should equal I2 at this point). Then a test (block **19E068**) is made to determine if the result of the comparison is EQUAL. If the result of the comparison is EQUAL, DISJOINT is returned to the calling process (block **19E070**). If the result of the comparison is not EQUAL, OVERLAP is returned to the calling process (block **19E072**).

Running a Class That Requests a Protected Operation

As discussed above with reference to FIGS. **13A–C**, a class (or other active content) is not allowed to run on a user's machine until a set of permissions are granted and the granted permissions are stored with the class (block **1318**). Permissions are active code (code that runs) that acts as an intermediary between the class and a protected operation on the host system. In an actual embodiment of the invention, permissions are modeled as objects that are stored together with the class for which they have been granted. The classes

and permissions may be either stored temporarily in the system RAM **125** and discarded following their use or the class and permissions may be persisted on the host system for use at a later time. In either circumstance, once the permissions are granted and attached to the class, there is no need to compare those permissions against those defined by the user for the system. This speeds access to the class and removes a potential impediment while it runs.

As illustrated in FIG. **22** when a class is run (block **2202**) the test is made to determine if the class has requested a protected operation (block **2204**). If a request for a protected operation is detected, a test (block **2206**) is made to determine whether the class has a granted permission for the protected action. If the class does not have a granted permission for the protected action the protected operation is not performed (block **2208**). If a test determines that logging has been enabled (block **2207**), the failure to perform the protected operation is noted (block **2210**) in a protected operation log. If logging has not been enabled, the process cycles back to wait for the next request for a protected operation (blocks **2202** and **2204**).

In an actual embodiment of the invention, the Java classes have methods which can initiate security checks when they are called. For example, the actual embodiment exposes a public class called 'java.io.File' which contains a public method called 'delete' which can be used to delete files. When this 'delete' method is called, a security check is initiated to ascertain that the caller(s) actually possess the correct permissions to delete the specified file.

If the class does have a granted permission for the protected action then a test (block **2212**) is made to verify that every class in the call chain also has the permission. FIG. **23** illustrates in flow diagram form how the test is conducted. The purpose of verifying that all classes in the call chain possess the permission requested is to prevent a security loophole often referred to as "luring". Luring occurs when a class that does not have a permission for a protected operation calls another class that does have the permission for the protected operation and tries to trick that class into doing something that the calling class is not permitted to do. The procedure illustrated in FIG. **23** screens for luring by verifying that every class that has called another class in a class call chain has the permission to perform the protected operation.

The first step in the verification test (FIG. **23**) is a test (block **2310**) to determine if there are any frames remaining in the call chain. Each frame represents a call from one class to another. For instance, Class A can call Class B which calls Class C which calls Class B. The call chain would be B C B A.

If there are remaining frames in the call chain (block **2310**), the immediately preceding caller in the call chain is retrieved (block **2312**). Then a test (block **2314**) is made to verify that the retrieved caller has the permission being analyzed. If the caller does not have the permission, the permission fails (block **2316**) and a NO decision (block **2316**) is returned (block **2318**). The returned NO decision prevents the protected operation from being performed (block **2208**; FIG. **22**).

Returning to FIG. **23**, if the caller has the permission (block **2314**), a test (block **2320**) is made to determine if the permission allows the requested action. If not, the permission fails an a not is returned. If the permission is allowed, the next stack frame is checked (block **2310**) and processing continues until either the permission fails (block **2316**) or the last stack frame is reached and processed. When the last stack frame is reached (block **2310**), the permission is verified and a YES (block **2322**) is returned (block **2318**). While an actual embodiment of the invention employs the "stack crawl" described in FIG. **23**, alternate methods could be used that will be apparent to one skilled in the art.

Returning to FIG. **22**, if the permission is verified (block **2212**), the protected operation is performed (block **2214**) and the process cycles back to wait for another protected operation request (blocks **2202** and **2204**).

While the preferred embodiment of the invention has been illustrated and described, it will be appreciated that various changes can be made therein without departing from the spirit and scope of the invention.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A computer-based method for providing security during a network browsing session between a client computer and a server computer, the method comprising:

receiving a document from the server computer at the client computer;

in response to receiving a request to perform an operation pertaining to the document, determining whether the requested operation is a protected operation;

if a protected operation is requested, determining a security setting corresponding to the protected operation and the server computer, wherein the security setting comprises:

at least one permission, the permission defining the access granted to the document on the client computer; and

selectively performing the protected operation based on the security setting.

2. A computer-implemented method for providing security on a host computer system by selectively restricting access by active content downloaded from a computer network to a protected system resource provided by the host computer system, the method comprising:

associating a permission with the protected system resource that regulates the access of the active content to the protected system resource;

providing a permission specification that defines how the permission associated with the protected system resource regulates access by the active content to the protected system resource;

detecting an attempt by the active content to access the protected system resource provided by the host computer system; and

granting the permission associated with the protected system resource to the active content if the attempt by the active content to access to the protected system resource falls within the permission specification.

3. The method of claim **2**, further comprising denying the permission associated with the protected system resource to the active content if the attempt by the active content to access to) the protected system resource does not fall within the permission specification.

4. The method of claim **2**, wherein if the permission associated with the protected system resource is granted to the active content, a granted permission for the system resource is associated with the active content such that access to the protected system resource is granted upon subsequent access attempts by the active content without again having to determine if the attempt by the active content to access to the protected system resource falls within the permission specification.

5. The method of claim **4**, further comprising persisting the granted permission with the active content.

6. The method of claim **5**, wherein the granted permission for the system resource is an object associated with the active content.

7. The method of claim **4**, wherein the active content has a procedure that calls another procedure forming a call chain, the method further comprising:

permitting the active content to access the protected system resource if each procedure in the call chain has the granted permission for the protected system resource; and

prohibiting the active content from accessing the protected system resource if each procedure in the call chain does not have the granted permission for the protected system resource.

8. The method of claim **2**, wherein the permission specification includes a parameter specification that defines a restriction on an action that may be taken when accessing the protected system resource.

9. The method of claim **8**, wherein the parameter specification includes a primitive specification that further defines the restriction on an action that may be taken when accessing the protected system resource.

10. The method of claim **9**, wherein the primitive specification is defined by an include/exclude pair of expressions.

11. The method of claim **9**, wherein primitive specification is defined by a inclusive Boolean expression.

12. The method of claim **9**, wherein the primitive specification is defined by an exclusive Boolean expression.

13. The method of claim **9**, wherein the primitive specification is defined by a numerical limit expression.

14. The method of claim **9**, wherein the primitive specification is defined by a regular expression.

15. The method of claim **14**, wherein the regular expression includes a wildcard expression.

16. The method of claim **15**, wherein the wildcard expression may be placed at any position in the regular expression.

17. The method of claim **2**, further comprising a security policy that includes a plurality of permission specifications with each permission specification defining how a corresponding permission associated with a protected system resource regulates access by the active content to the corresponding protected system resource.

18. The method of claim **17**, further comprising:

associating a digital signature with the security policy;

determining if a digital signature of the active content corresponds to the digital signature associated with the security policy; and

granting the permission associated with the protected system resource to the active content if the digital signature of the active content corresponds to the digital signature associated with the security policy and if the attempt by the active content to access the protected system resource falls within a permission specification in the security policy.

19. The method of claim **18**, wherein there are a plurality of security policies provided for the host computer system, the method further comprising granting the permission associated with the protected system resource to the active content if the attempt by the active content to access the protected system resource falls within a permission specification in the security policy associated with the digital signature borne by the active content.

20. The method of claim **17**, further comprising:

associating a network address with the security policy;

determining if a network address from which the active content originates corresponds to the network address associated with the security policy;

granting the permission associated with the protected system resource to the active content if the network address of the active content corresponds to the network address associated with the security policy and if the attempt by the active content to access the protected system resource falls within a permission specification in the security policy.

21. The method of claim **20**, wherein there are a plurality of security policies provided for the host computer system, the method further comprising granting the permission associated with the protected system resource to the active content if the attempt by the active content to access the protected system resource falls within a permission specification in the security policy associated with the network address from which the active content originates.

22. The method of claim **21**, further comprising:

associating a digital signature with the security policy;

determining if a digital signature of the active content corresponds to the digital signature associated with the security policy; and

granting the permission associated with the protected system resource to the active content if the digital signature of the active content corresponds to the digital signature associated with the security policy and if the attempt by the active content to access the protected system resource falls within a permission specification in the security policy.

23. The method of claim **22**, wherein there are a plurality of security policies provided for the host computer system, the method further comprising granting the permission associated with the protected system resource to the active content if the attempt by the active content to access, the protected system resource falls within a permission specification in the security policy associated with the digital signature borne by the active content.

24. The method of claim **23**, wherein the permission specification includes a parameter specification that defines a restriction on an action that may be taken when accessing the protected system resource.

25. The method of claim **24**, wherein the parameter specification includes a primitive specification that defines a set of items that may be acted on by an action that may be taken when accessing the protected system resource.

26. The method of claim **25**, further comprising a computer readable medium having computer-executable instructions for performing the preceding method.

27. The method of claim **25**, wherein the primitive specification is defined by an include/exclude pair of expressions.

28. The method of claim **25**, wherein primitive specification is defined by a inclusive Boolean expression.

29. The method of claim **25**, wherein the primitive specification is defined by an exclusive Boolean expression.

30. The method of claim **25**, wherein the primitive specification is defined by a numerical limit expression.

31. The method of claim **25**, wherein the primitive specification is defined by a regular expression.

32. The method of claim **31**, wherein the regular expression includes a wildcard expression.

33. The method of claim **32**, wherein the wildcard expression may be placed at any position in the regular expression.

34. A computer-implemented method for selectively restricting access by active content originating from a computer network to a protected operation on a host computer system, comprising:

associating a security zone with at least one network address on the computer network;

specifying a security policy for the security zone;

configuring a permission specification in the security policy, the permission specification defining a permitted access by active content having a network address associated with the security zone to the protected operation on the host computer system;

receiving a permission request associated with the active content, the permission request specifying a requested access that the active content seeks to the protected operation on the host computer system; and

granting a permission associated with the protected operation on the host computer system to the active content if the permission request falls within the permission specification.

35. The method of claim **34**, wherein a permission set is associated with the security zone, the permission set including a plurality of permission specifications.

36. The method of claim **35**, wherein the permission set is a granted permission set, the method further comprising:

comparing the permission request to a corresponding permission specification in the granted permission set; and

assigning a granted permission to the active content if the permission request falls within the permission specification included in the granted permission set.

37. The method of claim **36**, wherein the granted permission is only assigned to the active content if each of a plurality of permission requests fall within corresponding permission specifications included in the granted permission set.

38. The method of claim **36**, wherein the permission specification includes a parameter specification that defines a restriction on actions that the corresponding protected operation may perform on behalf of the active content.

39. The method of claim **38**, wherein the parameter specification includes a primitive specification that defines a set of items that the protected operation may take action upon on behalf of the active content.

40. A computer readable medium having computer-executable instructions for performing the method in any one of claims **34–39**.

41. The method of claim **39**, wherein the primitive specification is defined by an include/exclude pair of expressions.

42. The method of claim **39**, wherein primitive specification is defined by a inclusive Boolean expression.

43. The method of claim **39**, wherein the primitive specification is defined by an exclusive Boolean expression.

44. The method of claim **39**, wherein the primitive specification is defined by a numerical limit expression.

45. The method of claim **39**, wherein the primitive specification is defined by a regular expression.

46. The method of claim **45**, wherein the regular expression includes a wildcard expression.

47. The method of claim **46**, wherein the wildcard expression may be placed at any position in the regular expression.

48. The method of claim **35**, wherein the permission set is a denied permission set, the method further comprising:

comparing the permission request to a corresponding permission specification in the denied permission set; and

denying a granted permission to the active content if the permission request intersects the corresponding permission specification in the denied permission set.

49. The method of claim **48**, wherein the active content is denied any granted permission if any permission request associated with the active content intersects the denied permission set.

50. The method of claim **35**, wherein the permission set is a query permission set, the method further comprising:

comparing the permission request to a corresponding permission specification in the query permission set; and

querying for an instruction whether to assign a granted permission to the active content if the permission request falls within the corresponding permission specification included in the query permission set.

51. The method of claim **50**, wherein the security zone includes a denied permission set and the active content is denied any granted permission if any permission request associated with the active content intersects the denied permission set.

52. The method of claim **50**, wherein the granted permission is only assigned to the active content if each of a plurality of permission requests fall within each corresponding permission specifications included in the query permission set and the instruction whether to assign a granted permission to the active content is affirmative for each of the plurality of permission requests.

53. The method of claim **50**, wherein the permission specification includes a parameter specification that defines a restriction on an action that the corresponding protected operation may perform on behalf of the active content.

54. The method of claim **53**, wherein the parameter specification includes a primitive specification that defines a set of items that the protected operation may take action upon on behalf of the active content.

55. A computer readable medium having computer-executable instructions for performing the method in any one of claims **48–54**.

56. The method of claim **54**, wherein the primitive specification is defined by an include/exclude pair of expressions.

57. The method of claim **54**, wherein the primitive specification is defined by a inclusive Boolean expression.

58. The method of claim **54**, wherein the primitive specification is defined by an exclusive Boolean expression.

59. The method of claim **54**, wherein the primitive specification is defined by a numerical limit expression.

60. The method of claim **54**, wherein the primitive specification is defined by a regular expression.

61. The method of claim **60**, wherein the regular expression includes a wildcard expression.

62. The method of claim **61**, wherein the wildcard expression may be placed at any position in the regular expression.

63. A computer-implemented method for providing security on a host computer system to protect the host computer system against unauthorized access by foreign active content to a protected operation on the host computer system, comprising:

associating a security zone with at least one network address on the computer network;

defining a zone trust level for the security zone that includes at least one permission specification configured to reflect a level of trust that will be given to foreign active content from the network address associated with the security zone;

determining a permission request that specifies an access to the protected operation on the host computer system that the foreign active content desires in order to run on the host computer system; and

comparing the permission request to a corresponding permission specification included in the zone trust level

to determine a permission that controls access by the foreign active content to the protected operation on the host computer system.

64. The method of claim **63**, wherein there are a plurality of zone trust levels that are selectable to set a level of trust for the security zone.

65. The method of claim **64**, wherein one of the plurality of zone trust levels is a high security zone trust level that includes at least one permission specification configured to permit very restricted access to a corresponding protected operation provided by the host computer system.

66. The method of claim **65**, wherein one of the plurality of zone trust levels is a low security zone trust level that includes at least one permission specification configured to permit liberal access to a corresponding protected operation provided by the host computer system.

67. The method of claim **66**, wherein one of the zone trust level is a medium security zone trust level that includes at least one permission specification configured to permit moderately restricted access to a corresponding protected operation provided by the host computer system.

68. The method of claim **63**, wherein a plurality of permission specifications included in the zone trust level are grouped into permission sets.

69. The method of claim **68**, wherein the zone trust level includes a signed trusted permission set and the foreign active content is granted a permission to the protected operation on the host computer system commensurate with the permission request if the foreign active content has been digitally signed by a trusted publisher and the permission request falls within a corresponding permission specification in the signed trusted permission set.

70. The method of claim **69**, further comprising:

configuring a permission specification to restrict the access of the foreign active content to the protected operation on the host computer system in accordance with a predefined permission trust level;

determining a permission request that specifies an access to the system resource on the host computer system required by the foreign active content in order to operate on the host computer system; and

comparing the permission request to the permission specification to create the permission that regulates the access by foreign active content to the system resource on the host computer system.

71. The method of claim **70**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission, the method further comprising:

configuring the parameter specification to restrict the action in accordance with a predefined parameter trust level;

determining from the permission request a parameter request; and

comparing the parameter request to the parameter specification to determine a permitted action to include with the permission.

72. A computer readable medium having computer-executable instructions for performing the method in any one of claims **63**–**71**.

73. The method of claim **68**, wherein the zone trust level includes a signed untrusted permission set configured as a deny permission set and the foreign active content is denied a permission to the protected operation on the host computer system if the foreign active content has been digitally signed by an untrusted publisher and the permission request intersects with a corresponding permission specification in the signed untrusted permission set.

74. The method of claim **73**, further comprising:

configuring a permission specification to restrict the access of the foreign active content to the protected operation on the host computer system in accordance with a predefined permission trust level;

determining a permission request that specifies an access to the system resource on the host computer system required by the foreign active content in order to operate on the host computer system; and

comparing the permission request to the permission specification to create the permission that regulates the access by foreign active content to the system resource on the host computer system.

75. The method of claim **74**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission, the method further comprising:

configuring the parameter specification to restrict the action in accordance with a predefined parameter trust level;

determining from the permission request a parameter request; and

comparing the parameter request to the parameter specification to determine a permitted action to include with the permission.

76. A computer readable medium having computer-executable instructions for performing the method in any one of claims **73**–**75**.

77. The method of claim **68**, wherein the security zone trust level includes a signed untrusted permission set configured as a query permission set and a user is queried for an instruction whether to grant a permission to the protected operation on the host computer system if the foreign active content has been digitally signed by an untrusted publisher and the permission request falls within a corresponding permission specification in the signed untrusted permission set, the querying of the user comprising:

displaying a user interface requesting the instruction indicative of whether to grant a permission to the protected operation on the host computer system;

receiving the instruction indicative of whether to grant a permission to the protected operation on the host computer system; and

if the instruction indicative of whether to grant a permission to the protected operation on the host computer system is to grant the permission request, assigning a granted permission commensurate with to the permission request to the foreign active content; and

if the instruction indicative of whether to grant a permission to the protected operation on the host computer system is to deny the permission request, not assigning a granted permission to the foreign active content.

78. The method of claim **77**, further comprising:

configuring a permission specification to restrict the access of the foreign active content to the protected operation on the host computer system in accordance with a predefined permission trust level;

determining a permission request that specifies an access to the system resource on the host computer system required by the foreign active content in order to operate on the host computer system; and

comparing the permission request to the permission specification to create the permission that regulates the

access by foreign active content to the system resource on the host computer system.

79. The method of claim **78**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission, the method further comprising:

configuring the parameter specification to restrict the action in accordance with a predefined parameter trust level;

determining from the permission request a parameter request; and

comparing the parameter request to the parameter specification to determine a permitted action to include with the permission.

80. A computer readable medium having computer-executable instructions for performing the method of any one of claims **77–79**.

81. The method of claim **68**, wherein the zone trust level includes a unsigned permission set that specifies a set of default permissions that are granted to the foreign active content if the foreign active content has not been digitally signed by an active content publisher.

82. A computer-implemented method for protecting a system resource on a host computer system with a permission that regulates the access by foreign active content to the system resource on the host computer system, comprising:

providing a permission specification configured to restrict the access of the foreign active content to the system resource on the host computer system in accordance with a predefined permission trust level;

determining a permission request that specifies an access to the system resource on the host computer system required by the foreign active content in order to operate on the host computer system; and

comparing the permission request to the permission specification to create the permission that regulates the access by foreign active content to the system resource on the host computer system.

83. The method of claim **82**, wherein a plurality of predefined permission trust levels are user-selectable for the permission specification.

84. The method of claim **83**, wherein the plurality of predefined permission trust levels includes a high security permission trust level that is configured to provide very restricted access by the foreign active content to use the system resource on the host system.

85. The method of claim **83**, wherein the plurality of predefined permission trust levels includes a medium security permission trust level that is configured to provide a moderately restricted access by the foreign active content to use the system resource on the host system.

86. The method of claim **83**, wherein the plurality of predefined permission trust levels includes a low security permission trust level that is configured to provide a liberal access by the foreign active content to use the system resource on the host system.

87. The method of claim **83**, wherein the plurality of predefined permission trust levels includes a custom security permission trust level that is user-configurable to define an access by the foreign active content to use the system resource on the host system.

88. The method of claim **82**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission, the method further comprising:

configuring the parameter specification to restrict the action in accordance with a predefined parameter trust level;

determining from the permission request a parameter request; and

comparing the parameter request to the parameter specification to determine a permitted action to include with the permission.

89. A computer readable medium having computer-executable instructions for performing the method of any one of claims **81–88**.

90. The method of claim **89**, wherein a plurality of predefined parameter trust levels are user-selectable for the parameter specification.

91. The method of claim **90**, wherein the plurality of predefined parameter trust levels includes a high security parameter trust level that is configured to provide very restricted access by the foreign active content to use the system resource on the host system.

92. The method of claim **90**, wherein the plurality of predefined parameter trust levels includes a medium security parameter trust level that is configured to provide a moderately restricted access by the foreign active content to use the system resource on the host system.

93. The method of claim **90**, wherein the plurality of predefined parameter trust levels includes a low security parameter trust level that is configured to provide a liberal access by the foreign active content to use the system resource on the host system.

94. The method of claim **90**, wherein the plurality of predefined parameter trust levels includes a custom security parameter trust level that is user-configurable to define an access by the foreign active content to use the system resource on the host system.

95. The method of claim **94**, wherein the parameter specification includes a primitive specification that is defined by an include/exclude pair of expressions.

96. The method of claim **94**, wherein the parameter specification includes a primitive specification that is defined by a inclusive Boolean expression.

97. The method of claim **94**, wherein the parameter specification includes a primitive specification that is defined by an exclusive Boolean expression.

98. The method of claim **94**, wherein the parameter specification includes a primitive specification that is defined by a numerical limit expression.

99. The method of claim **94**, wherein the parameter specification includes a primitive specification that is defined by a regular expression.

100. The method of claim **99**, wherein the regular expression contains a wildcard expression.

101. The method of claim **99**, further wherein the regular expression contains a wildcard expression at any position in the regular expression.

102. A system for providing security on a computer system, comprising:

a permission object that controls the access of a computer program to a protected operation on the computer system;

a permission set including a permission specification that corresponds to the permission object and that defines an action that the permission object is allowed to take on behalf of the computer program;

a permission request that defines an action that the computer program requests that the permission object be allowed to take on behalf of the computer program; and

a security manager for constructing the permission object based on a comparison of the permission request to the permission specification.

103. The system of claim **102**, wherein the security manager compares the permission request to a corresponding permission specification included in a signed trusted permission set and the permission is constructed matching the permission request if the foreign active content has been digitally signed by a trusted publisher and the permission request falls within a corresponding permission specification in the signed trusted permission set.

104. The system of claim **103**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission and the security manager compares a parameter request to a parameter specification to construct the permission object with a parameter matching that specified in the permission request.

105. The system of claim **104**, wherein if the permission request is not digitally signed, the security manager constructs the permission object matching a corresponding permission specification in an a unsigned permission set.

106. The system of claim **105**, wherein if the permission request is digitally signed by an untrusted publisher, the security manager compares the permission request to a corresponding permission specification included in a signed untrusted permission set that has been configured as a deny permission set and the permission is not constructed if the permission request intersects a corresponding permission specification in the signed untrusted permission set.

107. The system of claim **106**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission and the security manager compares a parameter request to a parameter specification to determine whether to construct the permission object.

108. The system of claim **103**, wherein if the permission request is digitally signed by an untrusted publisher, the security manager compares the permission request to a corresponding permission specification included in a signed untrusted permission set that has been configured as a query permission set and the permission object is constructed matching the permission request only if a response to the query indicates that the permission object is allowed.

109. The system of claim **108**, wherein the permission specification includes a parameter specification that defines an action within the scope of the permission and the security manager compares a parameter request to a parameter specification to determine whether to construct the permission object.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1 Page 1 of 1
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

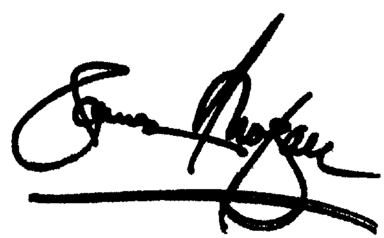
Column 77,
Line 19, "a unsigned" should read -- an unsigned --

Column 78,
Line 37, "a inclusive" should read -- an inclusive --

Column 79,
Line 18, "in an a" should read -- in an --

Signed and Sealed this

Sixth Day of May, 2003



JAMES E. ROGAN
Director of the United States Patent and Trademark Office

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1

Page 1 of 8

DATED : November 20, 2001

INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,
Item [57], **ABSTRACT**,
Line 11, “A protected operations” should read -- A protected operation --

Column 2,
Line 59, "that then to stored" should read -- then stored --

Column 3,
Line 1, "provides a way" should read -- provide a way --

Column 4,
Line 7, "and a "low" security" should read -- and "low" security --
Line 20, "contexts," should read -- contexts; --

Column 6,
Line 3, "package is identical" should read -- package are identical --
Line 49, "set.." should read -- set. --

Column 7,
Line 13, "of directional nature" should read -- of the directional nature --

Column 8,
Line 37, "a "Internet" should read -- an "Internet --
Line 54, "window dialog" should read -- dialog window --

Column 9,
Line 46, "lookup" should read -- look-up --
Line 61, "to of each class" should read -- to each class --

Column 10,
Line 1, "a users computer" should read -- a user’s computer --
Line 38, "Increasing," should read -- Increasingly, --

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1 Page 2 of 8
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column11,
Line 42, "network Process," should read -- network process, --
Line 62, "routines that helps" should read -- routines that help --

Column 12,
Line 2, "CD ROM" should read -- CD-ROM --
Line 22, "RAM **25,**" should read -- RAM **125,** --

Column 13,
Line 33, "that exclude" should read -- that excludes --
Line 53, "browser **204**" should read -- browser **204**, --

Column 15,
Line 39, "example, addition" should read -- example, in addition --

Column 16,
Line 60, "control **404**" should read -- control **404**, --

Column 17,
Line 29, "path/ie/plus/default.htm." should read -- path /ie/plus/default.htm. --
Line 42, "For instance" should read -- For instance, --

Column 19,
Line 15, "interface, **226**" should read -- interface **226** --

Column 20,
Line 21, "or whether to proceed" should read -- on whether to proceed --

Column 21,
Line 40, "set **618**" should read -- set **618**, --

Column 22,
Line 11, "1)." should read -- 1.) --
Lines 22-23, "(
www.microsoft.com)" should not break

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1 Page 3 of 8
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 24,
Line 10, "in a high security," should read -- in high security, --
Line 25, "is illustrated" should read -- are illustrated --
Line 35, "button **728** and **730**" should read -- buttons **728** and **730** --
Line 58, "interfaces **226**" should read -- interface **226** --
Line 67, "define a set" should read -- defines a set --

Column 25,
Line 11, "offer the user" should read -- offering the user --
Line 16, "content from that the user" should read -- content that the user --
Line 35, "Alternately, " should read -- Alternatively, --
Line 61, "(a)." should read -- (a.) --

Column 26,
Line 1, "set is shown" should read -- set are shown --
Line 28, "type is defined" should read -- type are defined --
Line 64, "set is represented" should read -- set are represented --
Line 67, "In effect" should read -- In effect, --

Column 28,
Line 2, "clients services" should read -- client services --
Line 53, "automatically been" should read -- automatically be --

Column 31,
Line 9, "in order to charge" should read -- in order to change --
Line 21, "by certificate holder," should read -- by a certificate holder, --
Line 29, "can not" should read -- cannot --

Column 33,
Line 23, "While actual" should read -- While an actual --

Column 34,
Line 3, "class intercepts" should read -- class intercept --
Line 19, "and to the untrusted signed" should read -- and the untrusted signed --
Line 46, "FIGS. 9*a-g*)" should read -- (FIGS. 9A-G) --

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1 Page 4 of 8
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 35,
Line 63, delete second occurrence of "has a"

Column 36,
Line 16, Table, line 4, "There is no items" should read -- There are no items --
Line 44, "**13***b*)" should read -- **13**B) --

Column 37,
Line 4, "from user defined set" should read -- from the user defined set --
Line 59, "are repeated." should read -- is repeated. --
Line 56, "**14***d*)" should read -- **14**D) --

Column 39,
Line 61, "described above" should read -- described above. --

Column 40,
Line 16, "(FIGS. **14**B)" should read -- (FIG. **14**B) --
Line 32, "FIGS. **14**B and" should read -- FIG. **14**B and --
Line 43, "The process for is" should read -- The process is --

Column 43,
Line 5,"**1640, 1640**" should read -- **1640, 1650** --

Column 47,
Line 5, "which is array" should read -- which is an array --
Lines 28-29, "PTRL-
IST**1**" should break -- PTR-
LIST**1** --

Column 48,
Lines 50-51, "PTRL-
IST**2**" should break -- PTR-
LIST**2** --

Column 49,
Line 19, "**18**I," should read -- **18**H, --
Line 32, "FIG. **18**X)." should read -- (FIG. **18**X). --

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 50,
Line 20, "starting at position" should read -- starting at the position --
Lines 29-30, "PTRL-
IST**2**" should break -- PTR-
LIST**2** --
Line 42, "Further" should read -- Further, --

Column 51,
Line 28, "CompareSingle-Components" should read -- CompareSingleComponents --
Lines 58-59, "CompareS-
ingleComponents" should break -- Compare-
SingleComponents --

Column 52,
Line 49, "lengthy" should read -- length --

Column 54,
Line 14, "intersecting a FoundIntersection" should read -- intersecting, a FoundIntersection --
Lines 34-35 "PTRL-
IST**1**" should break -- PTR-
LIST**1** --
Lines 38-39, "PTRL-
IST**1**" should break -- PTR-
LIST**1** --

Column 55,
Line 66, "process. i.e.," should read -- process; i.e., --

Column 56,
Line 7, "process shown" should read -- process is shown --
Line 46, "selected" should read -- selecting --
Line 57, "Separate-Component-Groups" should not be hyphenated
Line 67, "PTRL-" should read -- PTR- --

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

8PATENT NO. : 6,321,334 B1
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 57,
Line 1, "IST**2**" should read -- LIST**2** --

Column 59,
Line 6, "**180419**" should read -- **18O419** --
Line 58, "groups is the starting" should read -- groups in the starting --

Column 61,
Lines 13-14, "PTRL-
IST**1.** " should break --PTR-
LIST**1.** --
Lines 28-29, "PTRL-
IST**2**" should break -- PTR-
LIST**2** --
Line 54, "done first conducting" should read -- done by first conducting --

Column 63,
Line 13, "set is illustrated" should read -- set, is illustrated --

Column 64,
Line 11, "**19 A010**)" should read -- **19A010**) --

Column 65,
Line 25, "diagram from" should read -- diagram form --
Line 42, "E**2** COMPLETE" should read -- E**2**COMPLETE --
Line 56, "true," should read -- true; --

Column 66,
Line 23, "is illustrated" should read -- as illustrated --
Line 36, "true is" should read -- true if --
Line 50, "**19**H**090)**" should read -- **19**H**090)**, --

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1 Page 7 of 8
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 67,
Line 3, "Table **19-1** FIG." should read -- TABLE **19-1** in FIG. --
Line 47, "SetOpA-overlap" should read -- SetOpA-Overlap --
Line 59, "and a test" should read -- a test --

Column 69,
Line 3, "class and permissions" should read -- classes and permissions --
Line 64, "fails an a not" should read -- fails and a NO --

Column 70,
Line 57, "access to)" should read -- access to --

Column 71,
Lines 29, "a inclusive" should read -- an inclusive --

Column 72,
Line 3, "policy;" should read -- policy; and --
Line 35, "access, the" should read -- access the --
Line 52, "wherein primitive" should read -- wherein the primitive --
Line 53, "a inclusive" should read -- an inclusive --

Column 73,
Line 46, "wherein primitive" should read -- wherein the primitive --
Line 47, "a inclusive" should read -- an inclusive --

Column 74,
Line 38, "a inclusive" should read -- an inclusive --

Column 75,
Line 18, "level is" should read -- levels is --

Column 77,
Line 19, "a unsigned" should read -- an unsigned --

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,321,334 B1
DATED : November 20, 2001
INVENTOR(S) : M.S. Jerger et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 78,
Line 37, "a inclusive" should read -- an inclusive --

Column 79,
Line 18, "in an a" should read -- in an --

Signed and Sealed this

Twenty-fourth Day of June, 2003

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

Exhibit A-6

US 20020104014A1

(19) United States
(12) **Patent Application Publication** (10) **Pub. No.: US 2002/0104014 A1**
Zobel et al. (43) **Pub. Date: Aug. 1, 2002**

(54) **METHOD AND SYSTEM FOR CONFIGURING AND SCHEDULING SECURITY AUDITS OF A COMPUTER NETWORK**

(75) Inventors: **Robert David Zobel**, Atlanta, GA (US); **Timothy David Dodd**, Tucker, GA (US); **Sharon A. Millar**, Dawsonville, GA (US); **David Gerald Nesfeder JR.**, Suwanee, GA (US); **Christopher S. Singer**, Decatur, GA (US)

Correspondence Address:
Robert T. Neufeld, Esq.
KING & SPALDING
45th Floor
191 Peachtree Street, N.E.
Atlanta, GA 30303 (US)

(73) Assignee: **Internet Security Systems, Inc.**

(21) Appl. No.: **10/066,367**

(22) Filed: **Jan. 31, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/265,519, filed on Jan. 31, 2001.

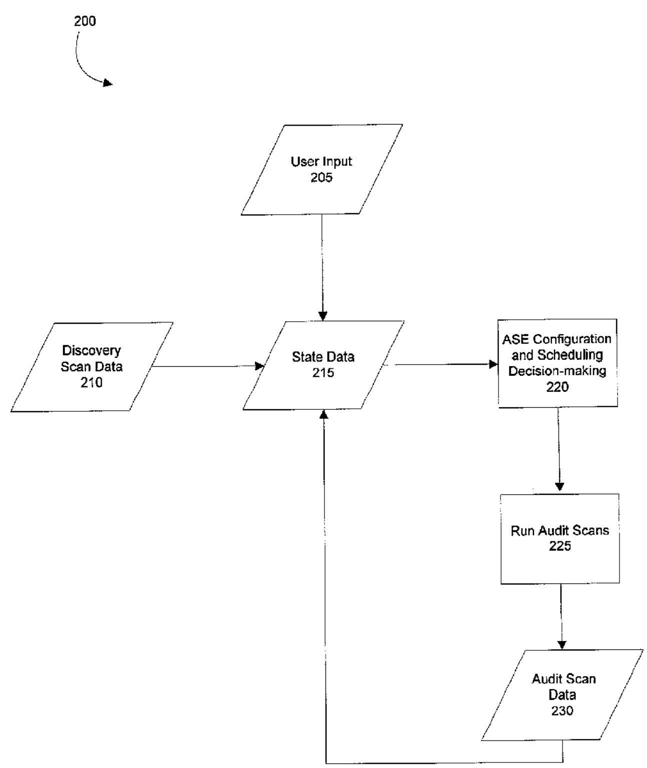
Publication Classification

(51) **Int. Cl.**[7] **G06F 11/30**
(52) **U.S. Cl.** **713/200**

(57) **ABSTRACT**

Managing the selection and scheduling of security audits run on a computing network. The computer network is surveyed by a security audit system to determine the function and relative importance of the elements in the network. Based on function and priority, a more thorough type of security audit is selected to run against each of the network elements by the security audit system. The security audit can also be automatically scheduled based on the information gathered from the survey. Once the system runs the security audit, a vulnerability assessment can be calculated for each element in the network. The vulnerability assessment can be presented in a format that facilitates interpretation and response by someone operating the system. The vulnerability assessment can also be used to configure and schedule future security audits.

DATA FLOW



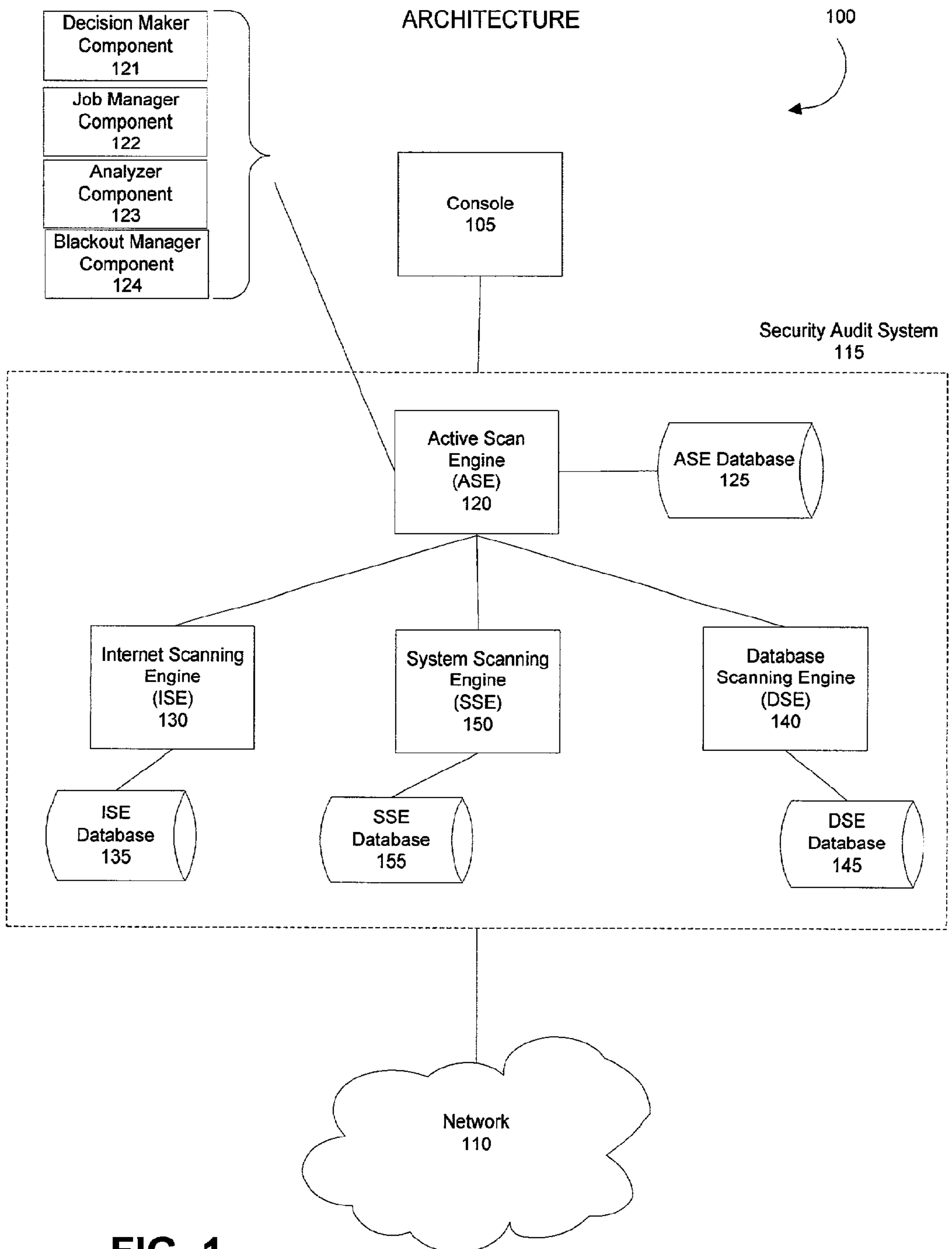

ARCHITECTURE
100
Decision Maker Component 121
Job Manager Component 122
Analyzer Component 123
Blackout Manager Component 124
Console 105
Security Audit System 115
Active Scan Engine (ASE) 120
ASE Database 125
Internet Scanning Engine (ISE) 130
System Scanning Engine (SSE) 150
Database Scanning Engine (DSE) 140
ISE Database 135
SSE Database 155
DSE Database 145
Network 110

FIG. 1

DATA FLOW

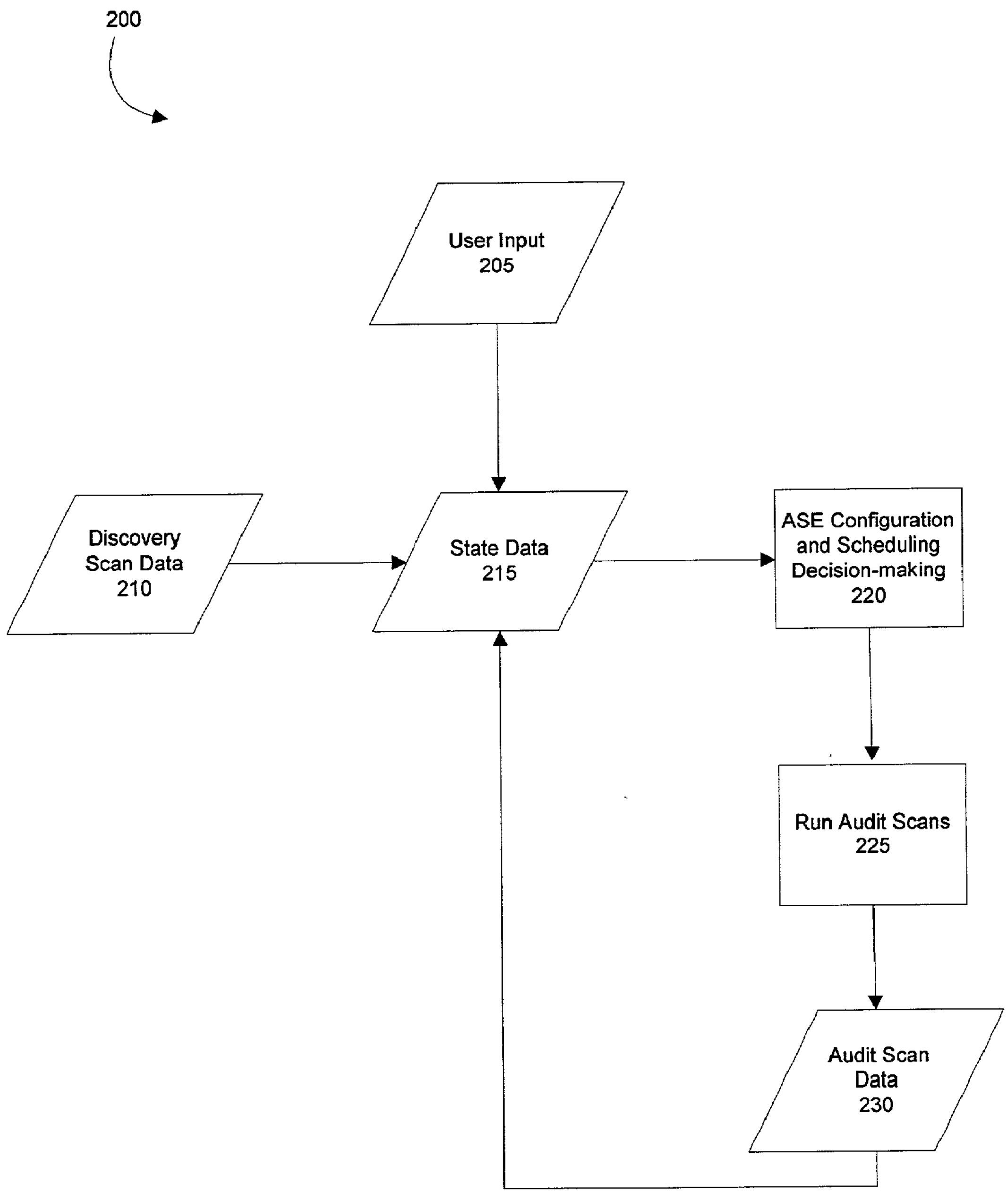


FIG. 2

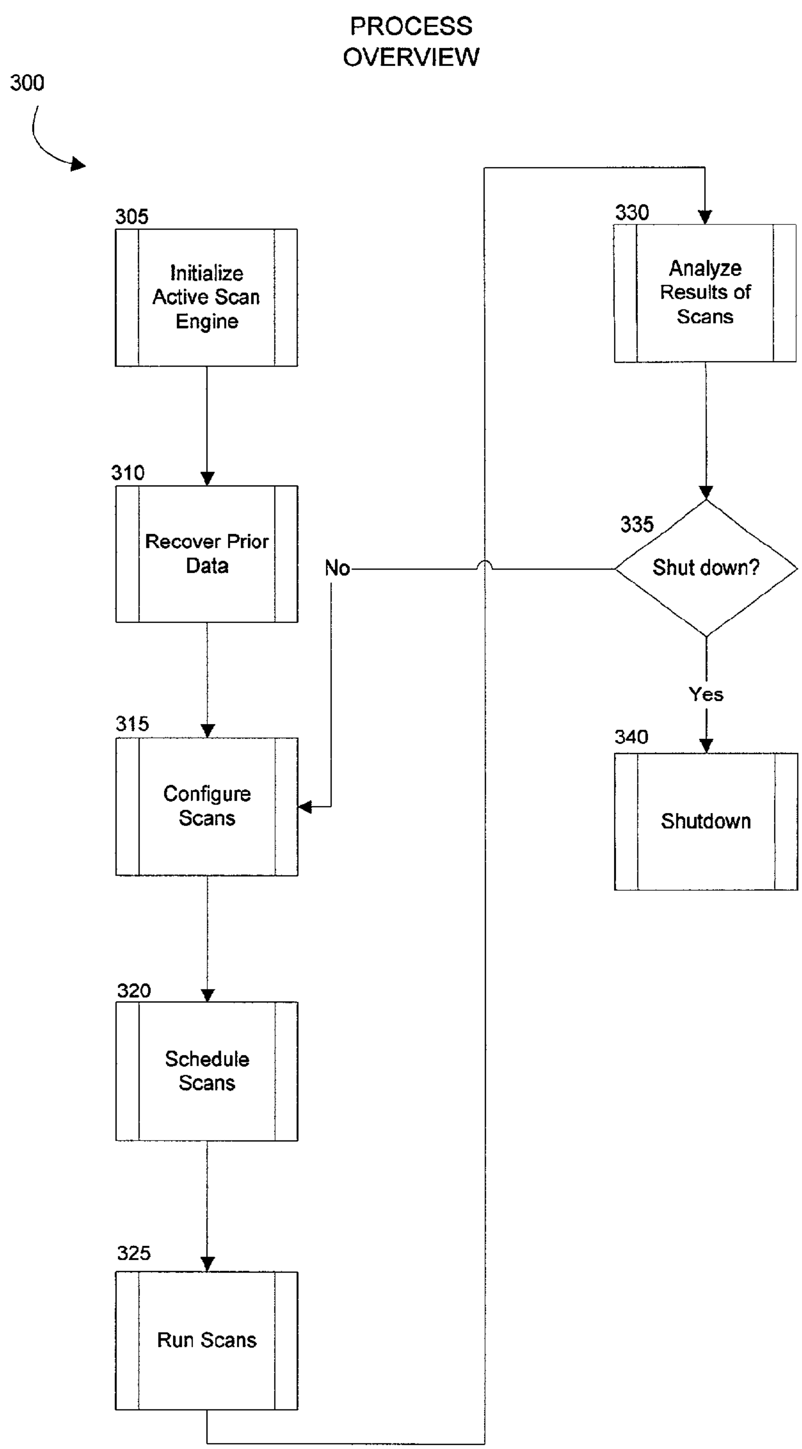

PROCESS
OVERVIEW
300
305
Initialize Active Scan Engine
310
Recover Prior Data
315
Configure Scans
320
Schedule Scans
325
Run Scans
330
Analyze Results of Scans
335
Shut down?
No
Yes
340
Shutdown

FIG. 3

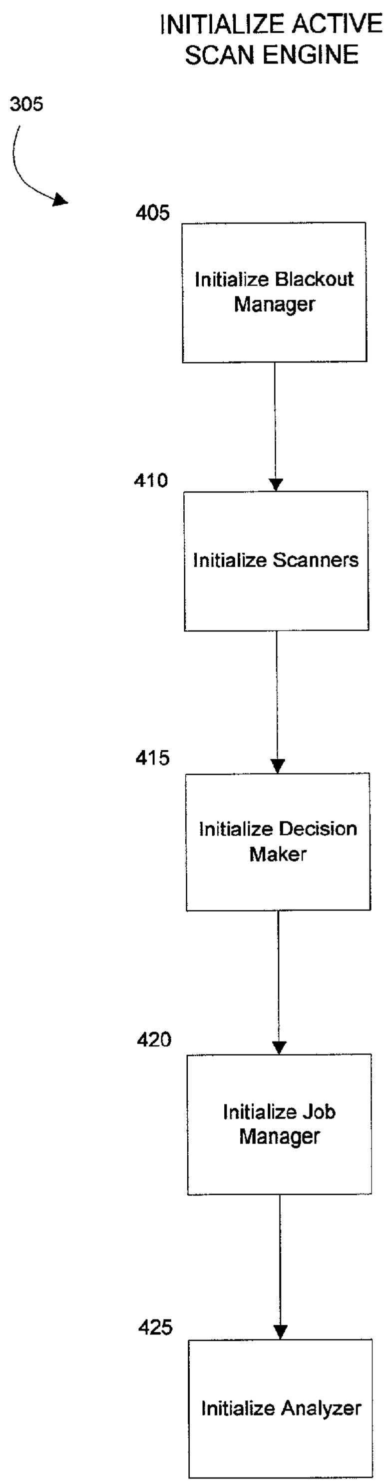

INITIALIZE ACTIVE SCAN ENGINE
305
405
Initialize Blackout Manager
410
Initialize Scanners
415
Initialize Decision Maker
420
Initialize Job Manager
425
Initialize Analyzer

FIG. 4

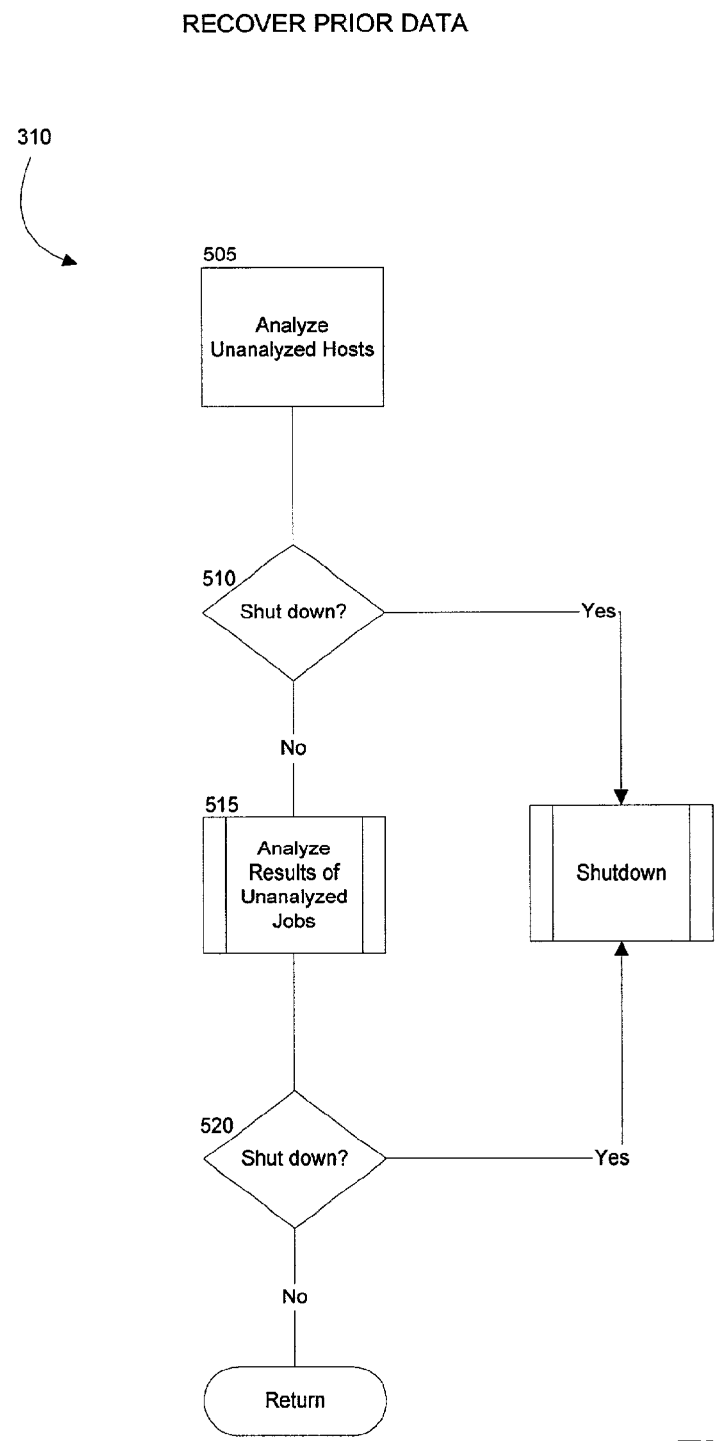

RECOVER PRIOR DATA
310
505
Analyze Unanalyzed Hosts
510
Shut down?
Yes
No
515
Analyze Results of Unanalyzed Jobs
Shutdown
520
Shut down?
Yes
No
Return

FIG. 5

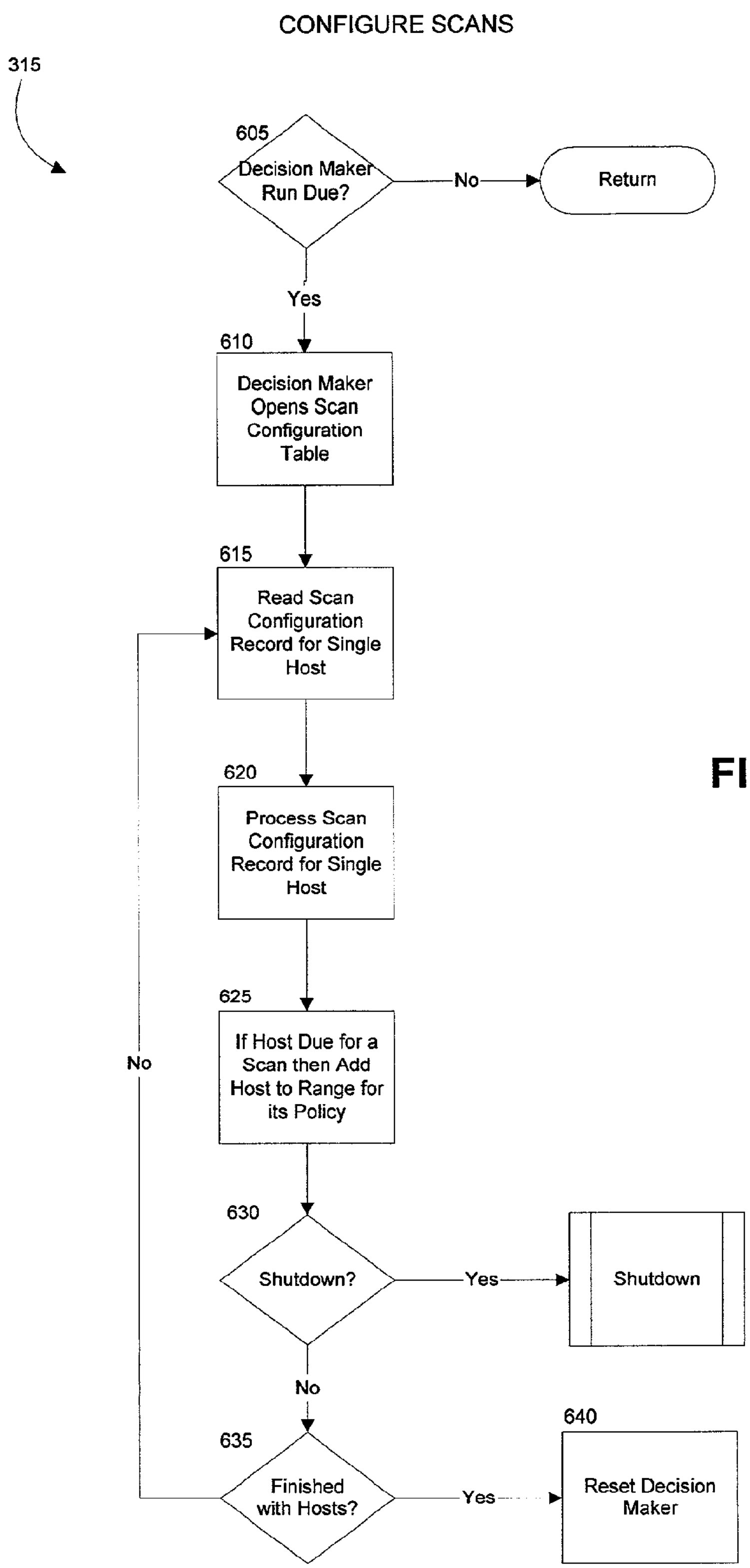

CONFIGURE SCANS
315
605
Decision Maker Run Due?
No
Return
Yes
610
Decision Maker Opens Scan Configuration Table
615
Read Scan Configuration Record for Single Host
620
Process Scan Configuration Record for Single Host
625
If Host Due for a Scan then Add Host to Range for its Policy
630
Shutdown?
Yes
Shutdown
No
635
Finished with Hosts?
No
Yes
640
Reset Decision Maker

FIG. 6A

CONFIGURE SCANS

Role	Asset Value	Scan Policy
UnixDesktop	3	L3UnixDesktop
UnixServer	4	L4UnixServer
UnixServer	5	L5UnixServer
UnixServer	6	L6UnixServer
NTDesktop	5	L5NTDesktop
NTServer	5	L5NTServer
SQLServer	3	L3SQLServer
SystemScanner	3	L3SystemScanner

Mapping from Host Role + Asset Value to a Scan Policy

FIG. 6B

Asset Value	Frequency
3	every 5 days
4	every 4 days
5	every 3 days
6	every 2 days
7	every day

Mapping from Asset Value to Scan Frequency

FIG. 6C

Entity	Host	Policy	Scan Frequency	First Job ID	Last Job ID
Internet Scanner	127.0.0.1	L3UnixWebServer	Every 5 days	1	3
Internet Scanner	135.1.2.3	L5NTWebServer	Every 2 days	2	3
Database Scanner	135.1.2.3	L3SQLServer	Every 5 days	3	4
System Scanner	127.0.0.1	L3Unix	Every 5 days	3	3

Scan Configuration Table

FIG. 6D

SCHEDULE SCANS

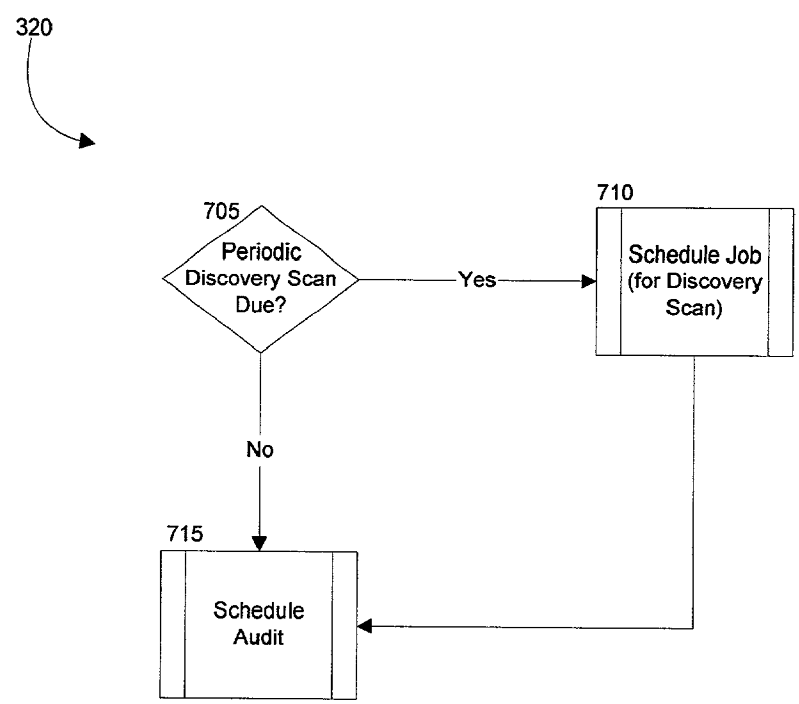


FIG. 7

SCHEDULE JOB

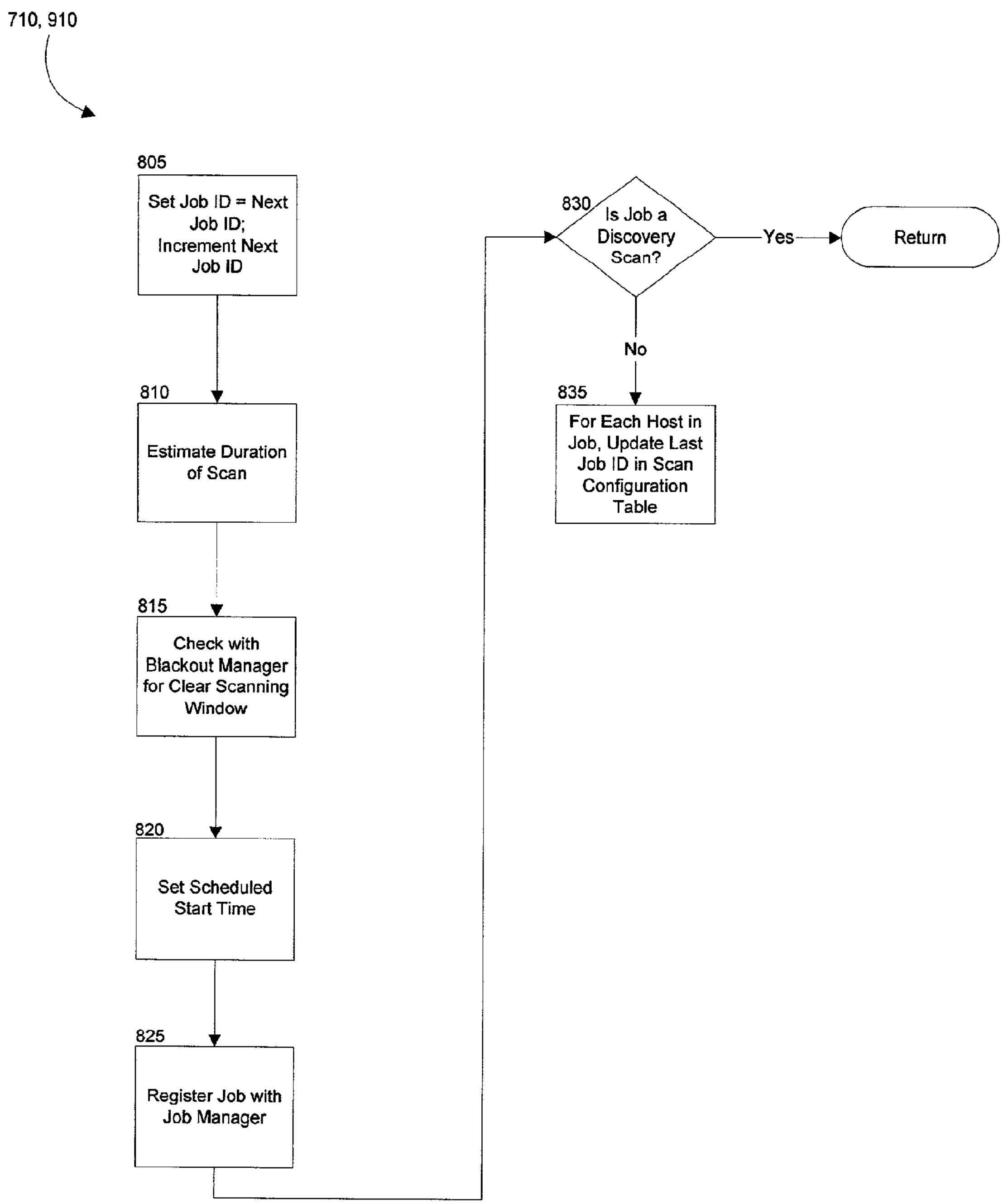


FIG. 8

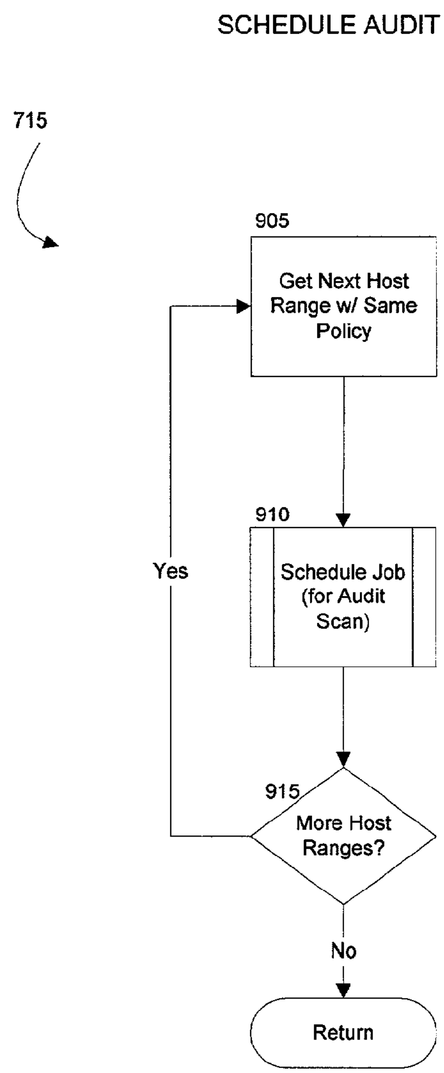

SCHEDULE AUDIT
715
905
Get Next Host Range w/ Same Policy
910
Schedule Job (for Audit Scan)
Yes
915
More Host Ranges?
No
Return

FIG. 9

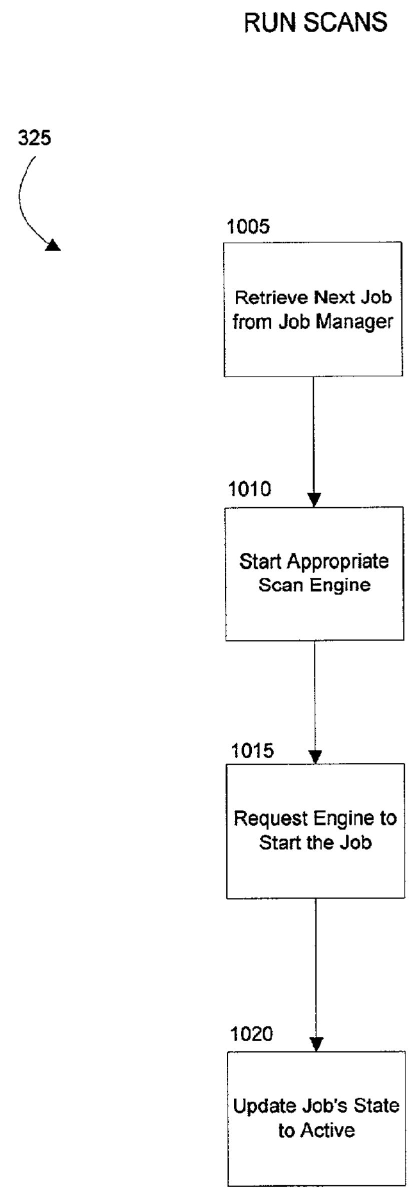

RUN SCANS
325
1005
Retrieve Next Job from Job Manager
1010
Start Appropriate Scan Engine
1015
Request Engine to Start the Job
1020
Update Job's State to Active

FIG. 10

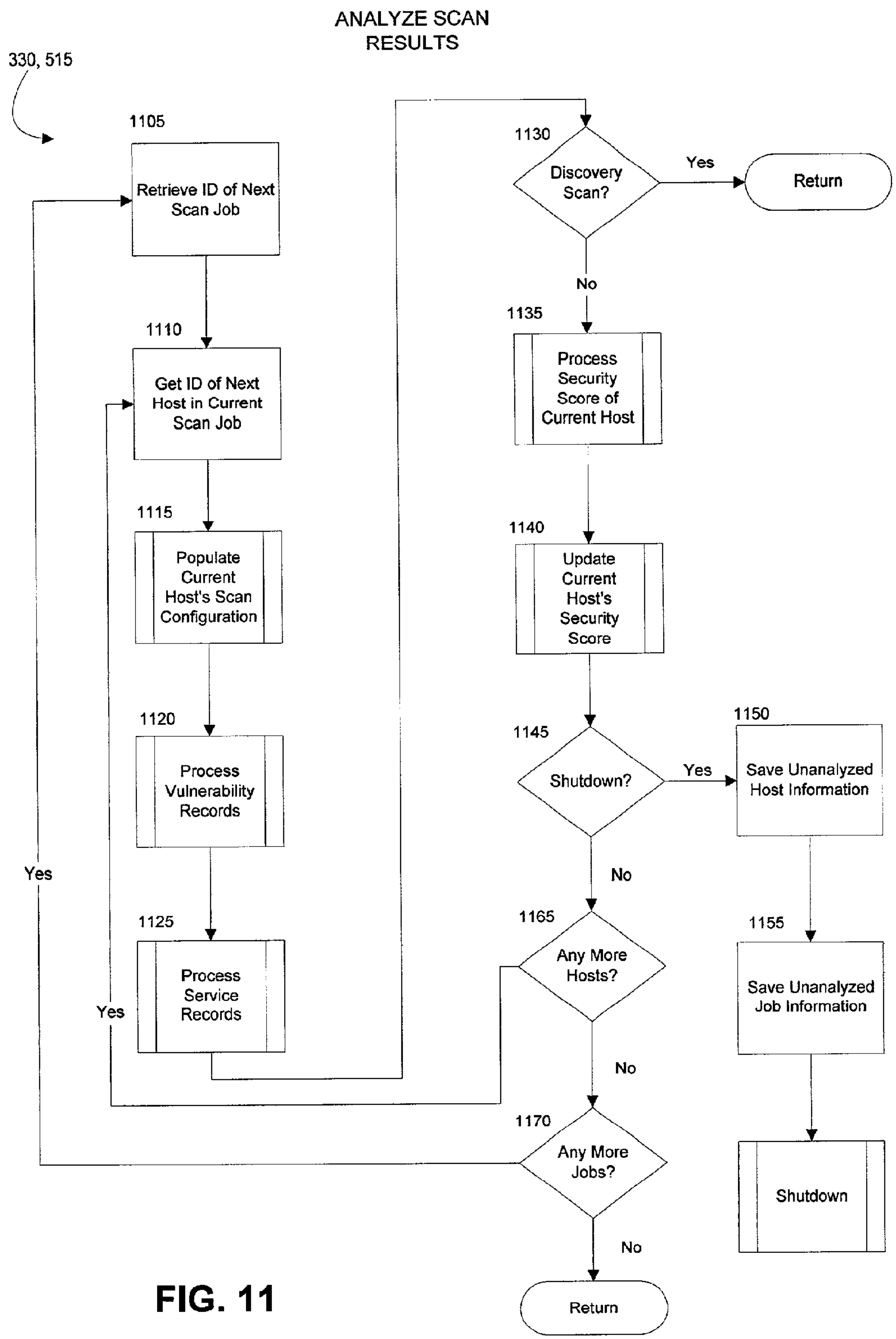

ANALYZE SCAN RESULTS
330, 515
1105
Retrieve ID of Next Scan Job
1110
Get ID of Next Host in Current Scan Job
1115
Populate Current Host's Scan Configuration
1120
Process Vulnerability Records
1125
Process Service Records
Yes
Yes
1130
Discovery Scan?
Yes
Return
No
1135
Process Security Score of Current Host
1140
Update Current Host's Security Score
1145
Shutdown?
Yes
1150
Save Unanalyzed Host Information
No
1165
Any More Hosts?
1155
Save Unanalyzed Job Information
No
1170
Any More Jobs?
Shutdown
No
Return

FIG. 11

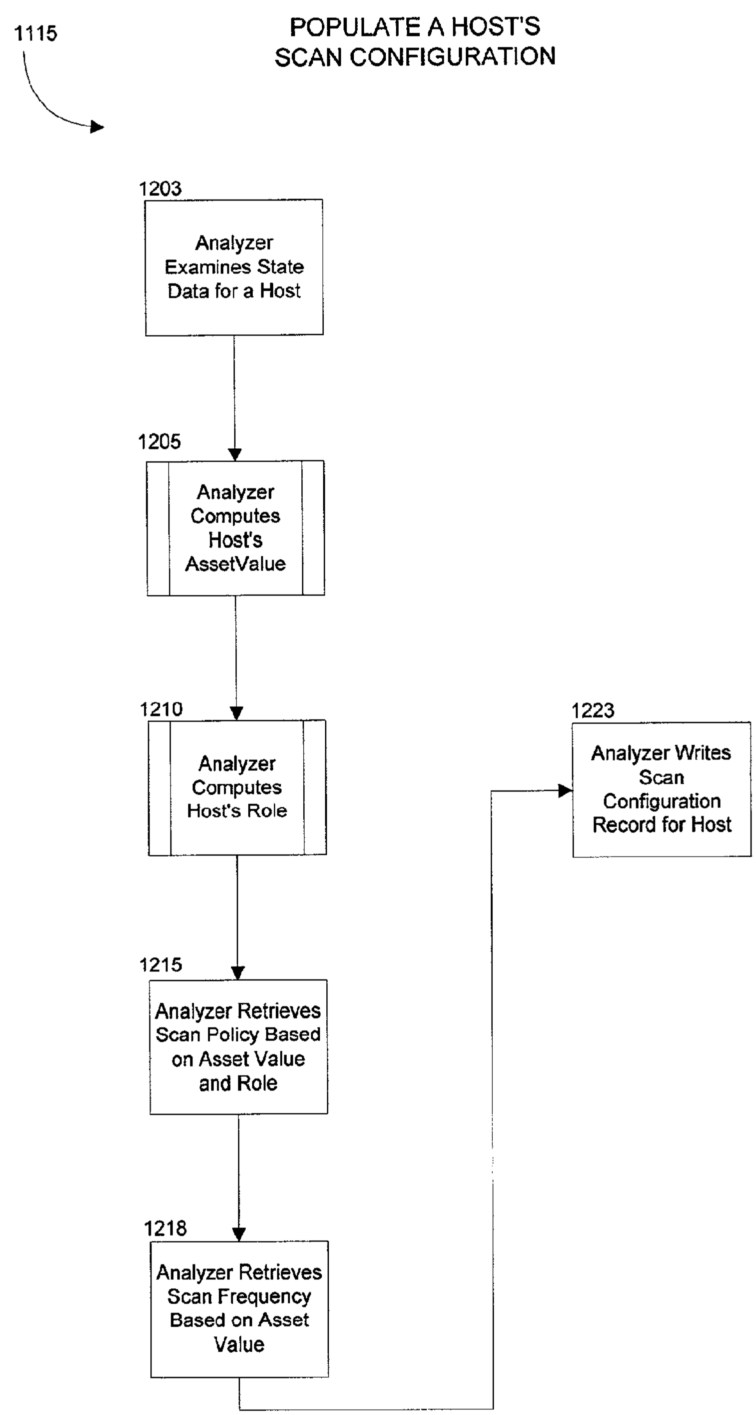

POPULATE A HOST'S SCAN CONFIGURATION
1115
1203
Analyzer Examines State Data for a Host
1205
Analyzer Computes Host's AssetValue
1210
Analyzer Computes Host's Role
1215
Analyzer Retrieves Scan Policy Based on Asset Value and Role
1218
Analyzer Retrieves Scan Frequency Based on Asset Value
1223
Analyzer Writes Scan Configuration Record for Host

FIG. 12

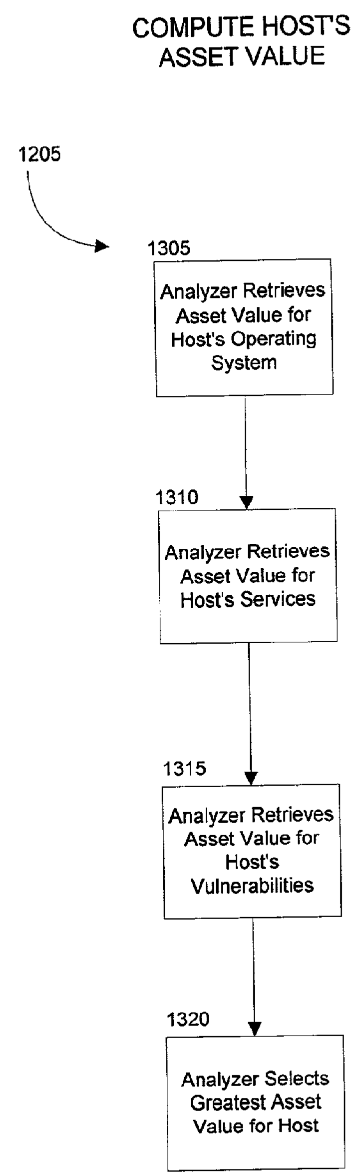

COMPUTE HOST'S ASSET VALUE
1205
1305
Analyzer Retrieves Asset Value for Host's Operating System
1310
Analyzer Retrieves Asset Value for Host's Services
1315
Analyzer Retrieves Asset Value for Host's Vulnerabilities
1320
Analyzer Selects Greatest Asset Value for Host

FIG. 13

COMPUTE HOST'S ROLE

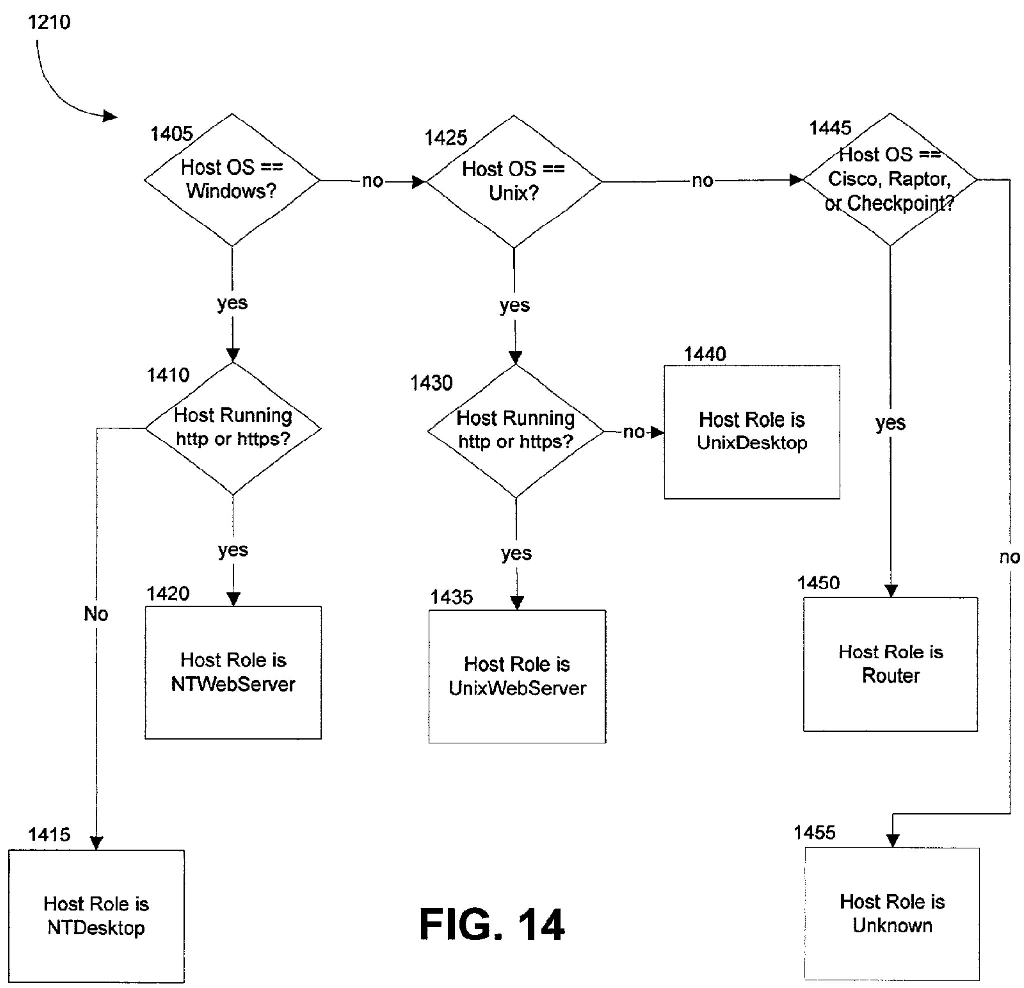


FIG. 14

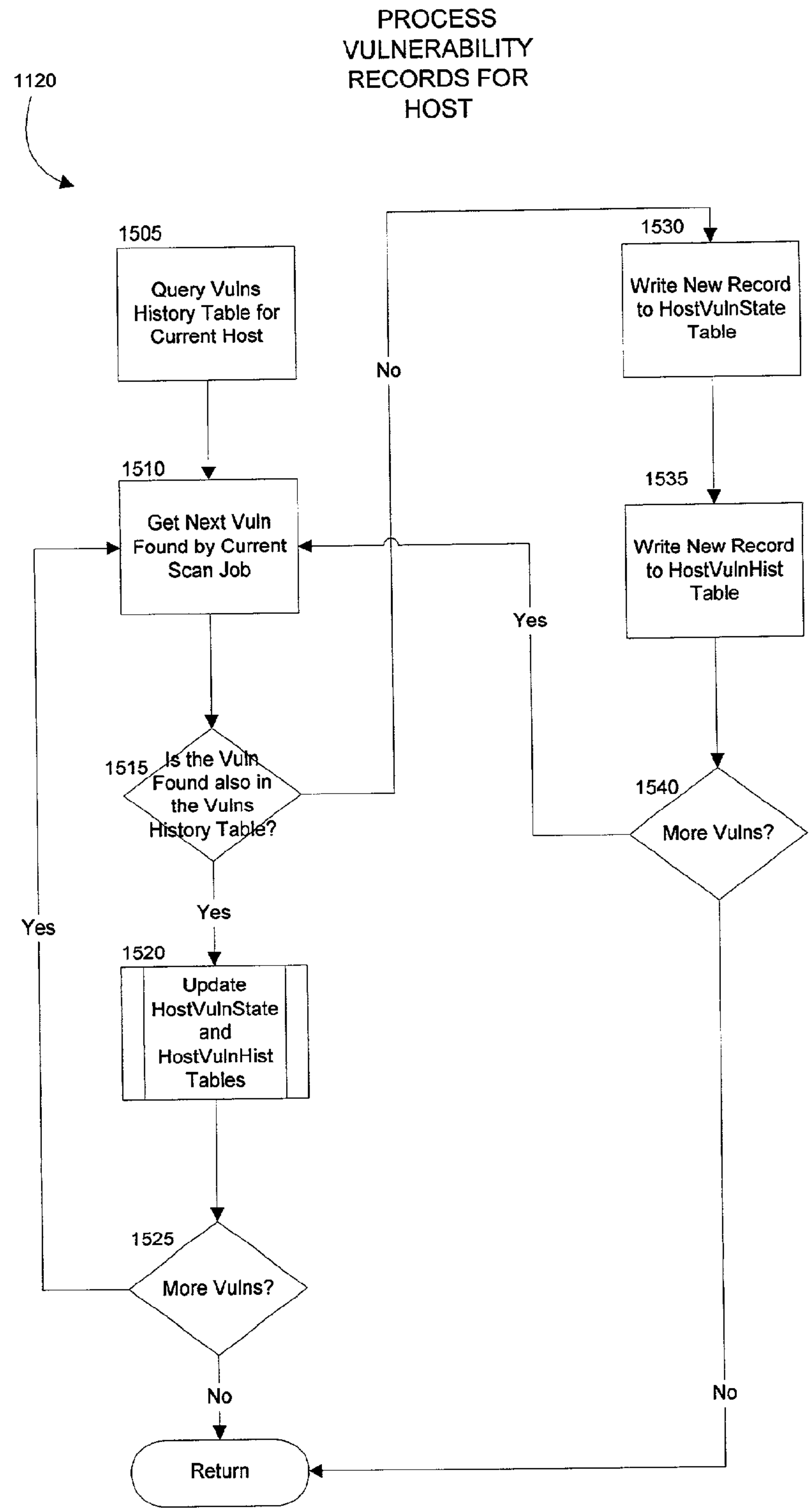

PROCESS VULNERABILITY RECORDS FOR HOST
1120
1505
Query Vulns History Table for Current Host
1510
Get Next Vuln Found by Current Scan Job
1515
Is the Vuln Found also in the Vulns History Table?
Yes
1520
Update HostVulnState and HostVulnHist Tables
1525
More Vulns?
Yes
No
Return
No
1530
Write New Record to HostVulnState Table
1535
Write New Record to HostVulnHist Table
1540
More Vulns?
Yes
No

FIG. 15

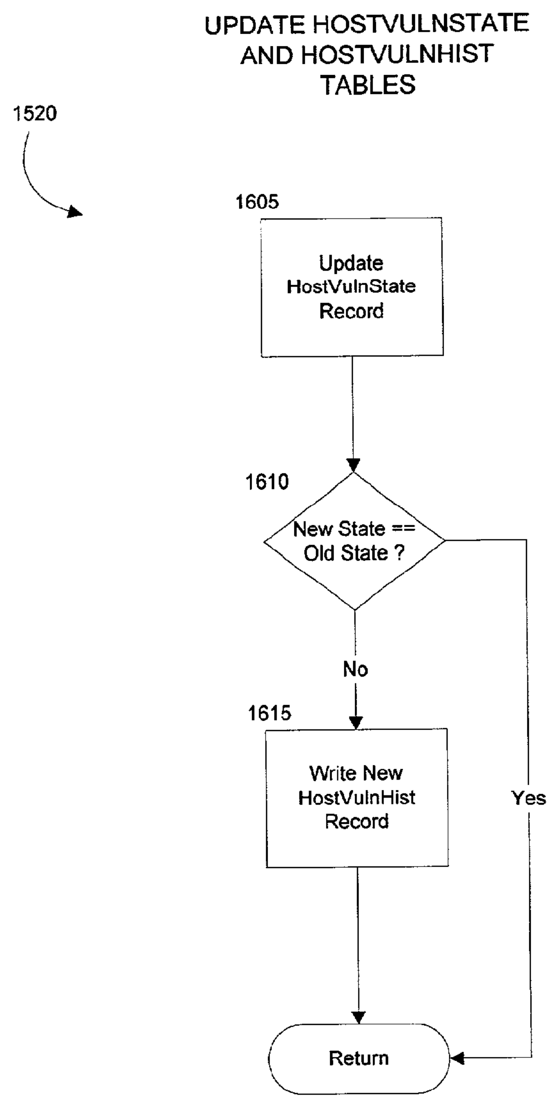

UPDATE HOSTVULNSTATE
AND HOSTVULNHIST
TABLES
1520
1605
Update
HostVulnState
Record
1610
New State ==
Old State ?
No
1615
Write New
HostVulnHist
Record
Yes
Return

FIG. 16

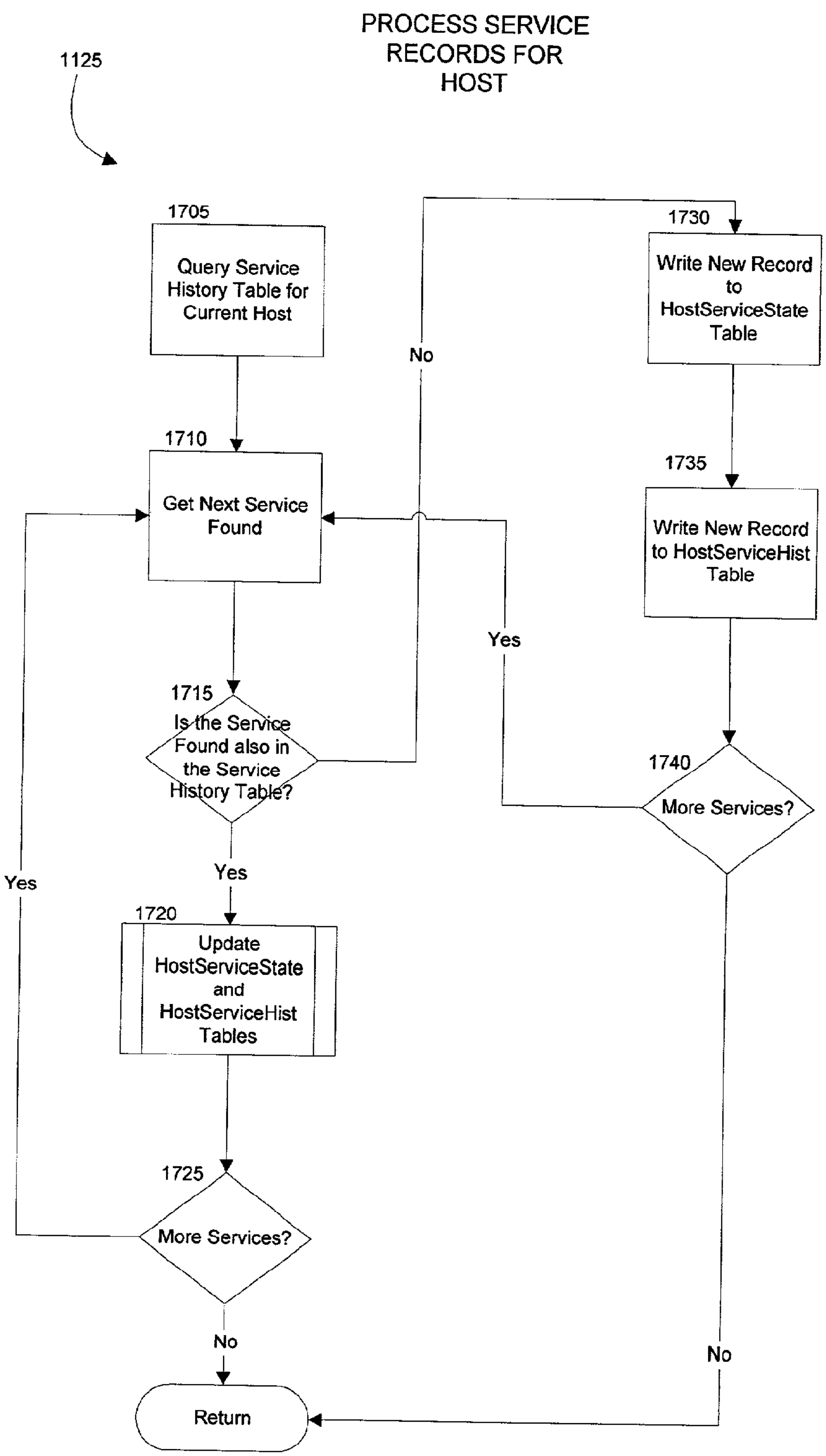

PROCESS SERVICE RECORDS FOR HOST
1125
1705
Query Service History Table for Current Host
1710
Get Next Service Found
1715
Is the Service Found also in the Service History Table?
Yes
1720
Update HostServiceState and HostServiceHist Tables
1725
More Services?
Yes
No
Return
No
1730
Write New Record to HostServiceState Table
1735
Write New Record to HostServiceHist Table
Yes
1740
More Services?
No

FIG. 17

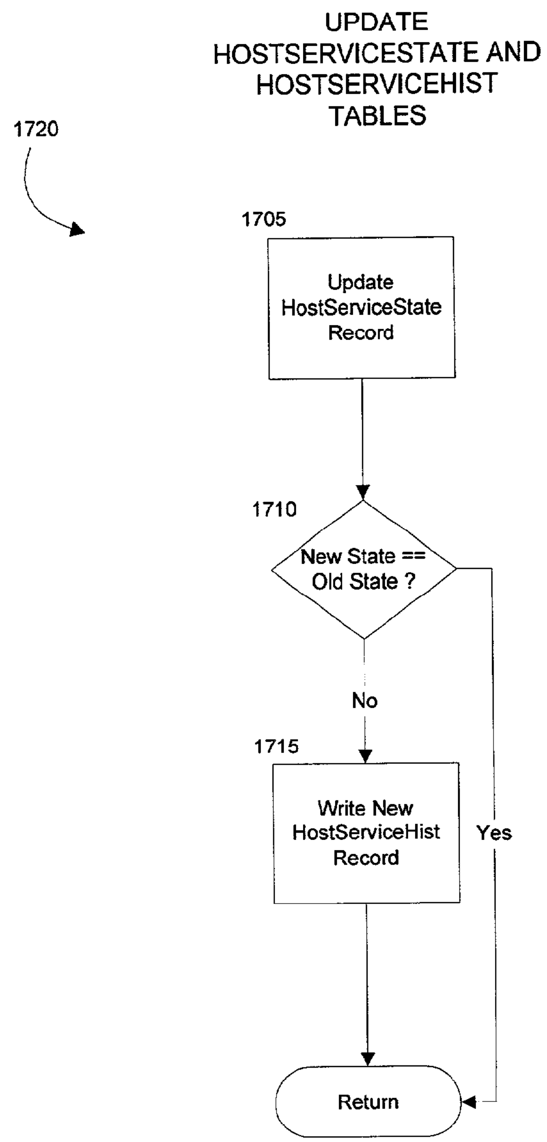

UPDATE HOSTSERVICESTATE AND HOSTSERVICEHIST TABLES
1720
1705
Update HostServiceState Record
1710
New State == Old State ?
No
Yes
1715
Write New HostServiceHist Record
Return

FIG. 18

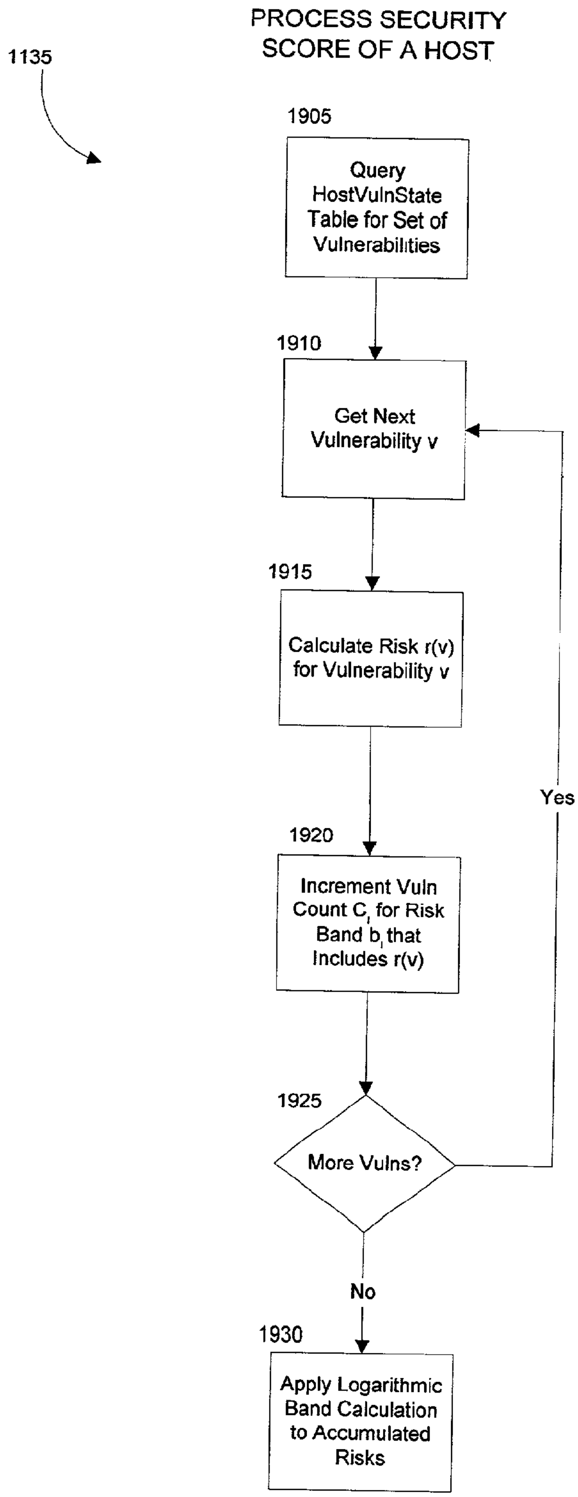

PROCESS SECURITY SCORE OF A HOST
1135
1905
Query HostVulnState Table for Set of Vulnerabilities
1910
Get Next Vulnerability v
1915
Calculate Risk r(v) for Vulnerability v
1920
Increment Vuln Count C_i for Risk Band b_i that Includes r(v)
1925
More Vulns?
Yes
No
1930
Apply Logarithmic Band Calculation to Accumulated Risks

FIG. 19

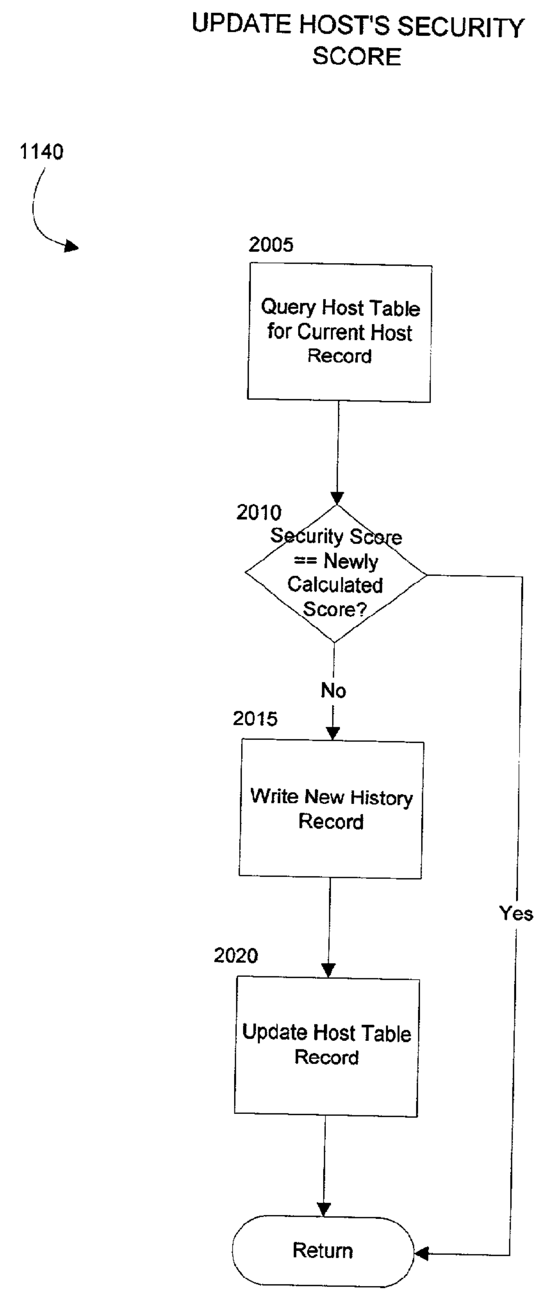

UPDATE HOST'S SECURITY SCORE
1140
2005
Query Host Table for Current Host Record
2010
Security Score == Newly Calculated Score?
No
Yes
2015
Write New History Record
2020
Update Host Table Record
Return

FIG. 20

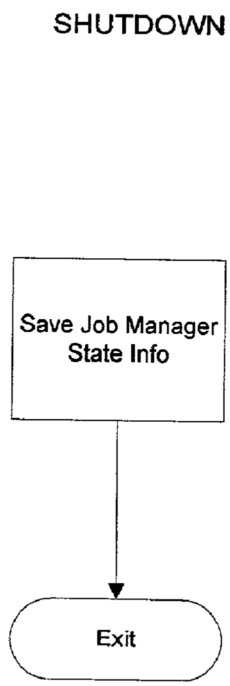


FIG. 21

METHOD AND SYSTEM FOR CONFIGURING AND SCHEDULING SECURITY AUDITS OF A COMPUTER NETWORK

PRIORITY AND RELATED APPLICATIONS

[0001] The present application claims priority to provisional patent application entitled, "Method and System for Configuring and Scheduling Security Audits of a Computer Network," filed on Jan. 31, 2001 and assigned U.S. application Ser. No. 60/265,519. The present application also references and incorporates herein a related U.S. non-provisional patent application entitled, "Method and System for Calculating Risk Associated with a Security Audit," filed concurrently herewith and having attorney docket number 05456.105036.

TECHNICAL FIELD

[0002] The present invention is generally directed to managing the security of a network. More specifically, the present invention facilitates the configuration and scheduling of security audits of machines in a distributed computer network.

BACKGROUND OF THE INVENTION

[0003] The security of computing networks is an increasingly important issue. With the growth of wide area networks (WANs), such as the Internet and the World Wide Web, people rely on computing networks to transfer and store an increasing amount of valuable information. This is also true of local area networks (LANs) used by companies, schools, organizations, and other enterprises. LANs are used by a bounded group of people in the organization to communicate and store electronic documents and information. LANs typically are coupled to or provide access to other local or wide area networks. Greater use and availability of computing networks produces a corresponding increase in the size and complexity of computing networks.

[0004] With the growth of networks and the importance of information available on the networks, there is also a need for better and more intelligent security. One approach to securing larger and more complex computer networks is to use a greater number and variety of security assessment devices. Security assessment devices can be used to evaluate elements in the network such as desktop computers, servers, and routers, and determine their respective vulnerability to attack from hackers. These network elements are commonly referred to as hosts and the terms "element" and "host" are used interchangeably herein. Security assessment devices can also be used more frequently to monitor the activity or status of the elements in a computing network.

[0005] One problem with increasing the number of security assessment devices and the frequency with which they are used is deciding which elements in the network need to be audited, how frequently they should be audited, and what checks need to be run. These are decisions that often involve a variety of complicated factors and they are decisions that in practicality cannot be made every time a security audit is conducted. Increased assessment also produces a corresponding increase in the amount of security data that must be analyzed. A network administrator that is overwhelmed with security data is unable to make intelligent decisions about which security vulnerabilities should be addressed first.

[0006] An additional problem associated with maintaining adequate network security is finding the time to conduct security audits. Security audits generally must be initiated by a security professional and can hinder or entirely interrupt network performance for several hours at a time. Furthermore, existing security assessment devices typically perform a variety of security scans on a machine, some of which may not be necessary. These unnecessary scans can translate into additional "down time" for the network.

[0007] In view of the foregoing, there is a need in the art for a system which will support the auditing of a distributed computing network. Specifically, a need exists to be able to automatically survey a network and determine the role and value of each element in the network. A further need exists to be able to assess the vulnerability of each element in the network. There is also a need to automatically schedule security auditing based on the vulnerability assessment of each element and to adjust future scheduling as audit data change. In this manner, those elements deemed to have the greatest risk can be monitored more closely. Finally, a need exists to be able to manage and present data pertaining to the survey, the vulnerability assessment, and the scheduling in a convenient graphical format.

SUMMARY OF THE INVENTION

[0008] The present invention satisfies the above-described needs by providing a system and method for scheduling and performing security audits in a distributed computing environment. Assessing the security of a relatively large or complex computer network can require hundreds of decisions about the types and timing of security checks. By facilitating the selection and scheduling of security audits, the present invention improves existing network security techniques. The present invention can identify the various elements in a distributed computing network and determine their role and relative importance. Using an element's role and relative importance, a more thorough security audit is chosen and scheduled to be run at an appropriate time. Information from the security audit can be used to calculate a security score and to modify the type and scheduling of future security audits. Security audit information can also be prioritized and presented to a user in a convenient format.

[0009] In one aspect, the present invention comprises a method for configuring and scheduling security scans of a computer network. A security audit system can conduct a discovery scan to identify elements that exist in a distributed computing network. Elements typically identified include, but are not limited to, desktop computers, servers, routers, and data storage devices. From the information collected during the discovery scan, the security audit system can determine the operating system and/or services associated with an element. The element's function and importance in the network can be used to configure an audit scan. An audit scan is a more thorough examination than a discovery scan and different types of audit scans involve different types of checks. The security audit system can schedule the selected audit scan to run at a time that will not interrupt the normal functioning of the computer network. The information collected during the audit scan can be used by the security audit system to calculate a security score for each element or group of elements. A security score is useful for identifying and prioritizing vulnerabilities that need to be remedied in the network.

[0010] In another aspect, the present invention provides a method for assessing the security of a network using a security audit system. The security audit system can receive information about elements in the network from an initial scan of the network. Using the information, the security audit system can select a more thorough audit scan to perform on a particular network. The selection of the audit scan can be based on the types of checks that need to be made on a particular element. The security audit system can also schedule the audit scan based on information collected during the initial scan. An element with greater importance or more serious vulnerabilities can be scanned more frequently than other elements in the network. Once the audit scan is performed, the security audit system receives more detailed information about the element and a security score can be computed for the element. The security score is useful in assessing the security of the network and prioritizing issues that need to be addressed.

[0011] For yet another aspect, the present invention further provides a security audit system for configuring and scheduling security scans of a computer network. The system comprises various types of scanning engines for running different scans and an active scan engine for coordinating the selection and scheduling of the different scans. The security audit system can conduct an initial scan to assess the functions and importance of various elements in the network. The initial scan provides information for deciding when to perform a more thorough audit scan and what type of audit scan to select. A console can also be coupled to the system for communicating information concerning the scans between a user and the security audit system.

[0012] These and other aspects of the invention will be described below in connection with the drawing set and the appended specification and claim set.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] **FIG. 1** is a block diagram illustrating an exemplary architecture for operating an embodiment of the present invention.

[0014] **FIG. 2** is a block diagram illustrating an exemplary data flow for a security audit system.

[0015] **FIG. 3** is a logic flow diagram illustrating an overview of the operating steps performed by a security audit system in accordance with an exemplary embodiment of the present invention.

[0016] **FIG. 4** is a logic flow diagram illustrating an exemplary process for initializing a scheduling module within a security audit system.

[0017] **FIG. 5** is a logic flow diagram illustrating an exemplary process for recovering prior data within a security audit system.

[0018] **FIG. 6A** is a logic flow diagram illustrating an exemplary process for configuring scans with a security audit system.

[0019] **FIGS. 6B, 6C**, and **6D** are exemplary tables associated with configuring scans.

[0020] **FIG. 7** is a logic flow diagram illustrating an exemplary process for scheduling scans with a security audit system.

[0021] **FIG. 8** is a logic flow diagram illustrating an exemplary process for scheduling specific scan jobs with a security audit system.

[0022] **FIG. 9** is a logic flow diagram illustrating an exemplary process for scheduling an audit scan with a security audit system.

[0023] **FIG. 10** is a logic flow diagram illustrating an exemplary process for running security scans with a security audit system.

[0024] **FIG. 11** is a logic flow diagram illustrating an exemplary process for analyzing scan results with a security audit system.

[0025] **FIG. 12** is a logic flow diagram illustrating an exemplary process for populating a host's Scan Configuration record with a security audit system.

[0026] **FIG. 13** is a logic flow diagram illustrating an exemplary process for computing a host's asset value with a security audit system.

[0027] **FIG. 14** is a logic flow diagram illustrating an exemplary process for computing a host's role with a security audit system.

[0028] **FIG. 15** is a logic flow diagram illustrating an exemplary process for processing vulnerabilities found on a host with a security audit system.

[0029] **FIG. 16** is a logic flow diagram illustrating an exemplary process for updating vulnerability state and history tables with a security audit system.

[0030] **FIG. 17** is a logic flow diagram illustrating an exemplary process for processing running services found on a host with a security audit system.

[0031] **FIG. 18** is a logic flow diagram illustrating an exemplary process for updating service state and history tables with a security audit system.

[0032] **FIG. 19** is a logic flow diagram illustrating an exemplary process for processing a host's security score with a security audit system **FIG. 20** is a logic flow diagram illustrating an exemplary process for updating a host's security score with a security audit system **FIG. 21** is a logic flow diagram illustrating an exemplary process for shutting down a security audit system.

DETAILED DESCRIPTION OF THE EXEMPLARY EMBODIMENTS

[0033] The present invention supports the automated assessment of the security risks of a computing network. Specifically, the present invention allows a security auditing system to collect initial information about the identity and importance of elements in a computing network. Using this initial information, the invention then provides for automatic selection and scheduling of security audit scans to be performed on the network elements. A user can provide parameters, if so desired, as to when to schedule audit scans and what types of audit scans to run. Taking the information collected from the audit scan, the auditing system can compute a security score for a network element based on its vulnerability and importance. The security score can be presented to the user in a manageable format to facilitate

interpretation and response. The user may use the security score as a basis for adjusting the scheduling and configuration of future audit scans.

[0034] Although the exemplary embodiments will be generally described in the context of software modules running in a distributed computing environment, those skilled in the art will recognize that the present invention also can be implemented in conjunction with other program modules for other types of computers. In a distributed computing environment, program modules may be physically located in different local and remote memory storage devices. Execution of the program modules may occur locally in a standalone manner or remotely in a client/server manner. Examples of such distributed computing environments include local area networks of an office, enterprise-wide computer networks, and the global Internet.

[0035] The detailed description that follows is represented largely in terms of processes and symbolic representations of operations in a distributed computing environment by conventional computer components, including database servers, application servers, mail servers, routers, security devices, firewalls, clients, workstations, memory storage devices, display devices and input devices. Each of these conventional distributed computing components is accessible via a communications network, such as a wide area network or local area network.

[0036] The processes and operations performed by the computer include the manipulation of signals by a client or server and the maintenance of these signals within data structures resident in one or more of the local or remote memory storage devices. Such data structures impose a physical organization upon the collection of data stored within a memory storage device and represent specific electrical or magnetic elements. These symbolic representations are the means used by those skilled in the art of computer programming and computer construction to most effectively convey teachings and discoveries to others skilled in the art.

[0037] The present invention also includes a computer program that embodies the functions described herein and illustrated in the appended flow charts. However, it should be apparent that there could be many different ways of implementing the invention in computer programming, and the invention should not be construed as limited to any one set of computer program instructions. Further, a skilled programmer would be able to write such a computer program to implement the disclosed invention based on the flow charts and associated description in the application text, for example. Therefore, disclosure of a particular set of program code instructions is not considered necessary for an adequate understanding of how to make and use the invention. The inventive functionality of the claimed computer program will be explained in more detail in the following description in conjunction with the remaining figures illustrating the program flow.

[0038] Referring now to the drawings, in which like numerals represent like elements throughout the several figures, aspects of the present invention and the preferred operating environment will be described.

[0039] **FIG. 1** illustrates various aspects of an exemplary computing environment in which an embodiment of the present invention is designed to operate. Those skilled in the art will appreciate that **FIG. 1** and the associated discussion are intended to provide a brief, general description of the computer network resources in a representative distributed computer environment including the inventive security audit system. The architecture comprises a console **105** and a security audit system **115** which are used to configure and schedule security audits of a network **110**. The console **105** communicates information about the current security state of the network **110** to a user. The console **105** typically comprises a graphical user interface for presenting and managing data in a convenient format for the user. The console **105** is also operable for receiving information from the security audit system **115** and allowing control of the security audit system **115**. The security audit system **115** comprises an active scan engine **120** and one or more other scan engines. In the exemplary embodiment illustrated in **FIG. 1**, the active scan engine **120** is coupled to an Internet scanning engine **130**, a system scanning engine **150**, and a database scanning engine **140**. Each of these scan engines illustrated in **FIG. 1** is coupled to a corresponding database.

[0040] The active scan engine's **120** primary task is acquiring and maintaining current data about the configuration and security posture of the network **110**. The active scan engine **120** utilizes the subsidiary scan engines **130**, **140** and **150** as a means for gathering information about the network **110**. The network **110** typically comprises elements such as desktop computers, routers, and various servers. The active scan engine **120** is responsible for coordinating the configuration, scheduling, and running of scans of these elements found in the network **110**. Typically, the active scan engine **120** is continuously running so that the scheduled scans can be run at their designated times, and the resultant data processed in a timely manner.

[0041] **FIG. 2** illustrates a flow chart diagram of an exemplary flow of data for an embodiment of the present invention. Beginning with the user input **205**, this represents data that a user may input at the console **105** to be used by the security audit system **115** in configuring and scheduling scans. User input data **205** can include a specific network range in which to conduct scans, fixed asset and vulnerability values, and blackout periods during which security scans should not be run. The user input **205** is combined with data recovered by the discovery scan **210**. The discovery scan is an initial scan of the network **110** run by the security audit system **115**. The discovery scan is used to identify the elements on the network **110** and to assign asset values to them. An asset value is typically an arbitrary value assigned based on the importance of an element relative to other elements in a network. By identifying the function and asset value of each element on the network, the security audit system **115** can configure and schedule further scans to run on the network **110**.

[0042] The user input **205** and the discovery scan data **210** are combined to formulate state data **215** describing each of the elements in the network **110**. In step **220**, the active scan engine **120** makes decisions regarding the types of scans to be run and when they will be run on the network **110**. Ultimately, scheduled audit scans will be run against each of the elements on the network **110** in step **225**. The audit scan involves a more thorough examination of a network element than the discovery scan. The audit scan data **230** is collected and fed back into the accumulated state data **215** describing

each element on the network **110**. The feedback mechanism shown in **FIG. 2** enables the security audit system **115** to adjust the configuring and scheduling of future audit scans. The exemplary method illustrated in **FIG. 2** allows the most important or most vulnerable elements of the network **110** to be given the highest priority in conducting security audits.

[**0043**] An exemplary process overview for operating the security audit system **115** is illustrated in **FIG. 3**. When the security audit system **115** first starts up, a scheduling module with the active scan engine **120** is initialized as shown in step **305** of **FIG. 3**. The initialization process involves adjusting several modules, which will be discussed in connection with **FIG. 4**. In step **310**, the active scan engine **120** recovers prior data from previous incompletely processed scans. The need to recover prior data results from the active scan engine **120** being shut down when there are scan results pending processing. When a shutdown occurs, any data necessary to recover the current state of the active scan engine **120** is saved to the ASE database **125**. Upon startup, the active scan engine **120** recovers and processes any incompletely analyzed data as further illustrated in **FIG. 5**.

[**0044**] In step **315**, the active scan engine **120** configures the scans that are to be run on the network **110**. The configuring of scans, discussed in greater detail in connection with **FIG. 6**, involves determining the type of scan to run on a network host based on its function and asset value. Once it is decided what type of scans will be run in step **315**, the active scan engine **120** chooses when to conduct scans on network elements in step **320**. The security audit system **115** can make many of the configuration and scheduling decisions that would ordinarily have to be made by the user.

[**0045**] In step **325**, the security audit system **115** runs the scheduled scans on the network **110**. The first time that a security audit system **115** scans the network **110** it will conduct a discovery scan to identify network elements and their function. The discovery scan collects information for use in subsequent configuring and scheduling of more thorough audit scans. After these scans are performed, the active scan engine **120** analyzes the data that are collected in step **330**. The analysis of the data can be used to readjust the configurations and scheduling of scans by returning to step **315**. Alternatively, the user can shut down the security audit system **115** in step **340**. The security audit system **115** performs tasks asynchronously from the user's perspective. When engaged in a potentially time-consuming task such as the analysis of scan results (step **330**), the active component periodically checks for a user-initiated shutdown signal. This allows the security audit system **115** to shut down in a timely manner even when engaged in lengthy tasks. As mentioned above, if a shutdown occurs when the security audit system **115** is engaged upon one or more tasks, sufficient state information is stored to allow for recovery upon reactivation. The foregoing steps are merely an exemplary embodiment of how to use the security audit system **115**. In an alternative embodiment of the invention, the foregoing steps may be performed in a different order or certain steps may be skipped entirely.

[**0046**] **FIG. 4** illustrates an exemplary method for initializing the active scan engine **120**. In step **405**, the active scan engine **120** loads the scan blackout schedule, initializing the blackout manager **124**. The scan blackout schedule contains time periods during which scans are not to be performed on the network **110**. The scan blackout schedule is typically determined by a user and entered using the console **105**. In step **410**, the various scanners **130**, **140**, and **150** are initialized so they will be ready to perform scans on the network **110**. The exemplary embodiment described herein discusses three types of scanning engines that can be used to perform a security audit on a network. An alternative embodiment of the present invention may employ a selected subset of these scan engines or other types of scan engines.

[**0047**] The Internet scanning engine **130** is a network scanning tool used to conduct the initial discovery scans performed on the network. The Internet scanning engine **130** can also be used to identify security vulnerabilities that exist across an entire network. The database scanning engine **140** performs audits of database servers identified by the Internet scanning engine **130** during the discovery scan. The system scanning engine **150** is a security auditing tool comprising software that is generally installed on individual hosts in the network. The system scanning engine **150** is typically installed on desktop computers, servers, and routers that have at least a specific asset value. Because they execute on the local host, the system scanning engine **150** is able to detect vulnerabilities that may be unidentifiable by the Internet scanning engine **130**. The active scan engine **120** works with the system scanning engine **150** to configure and schedule particular scans to be run on network elements.

[**0048**] In steps **415**, **420**, and **425**, components of the active scan engine **120** are initialized in preparation for conducting scans. Identified in step **415**, the decision maker component **121** receives the results of prior audit and discovery scans and determines which audit scans to run on which elements and when to run them. The job manager component **122**, initialized in step **420**, receives instructions from the decision maker component **121** and monitors what audit scans have been scheduled, when the audit scans have been scheduled, and whether the audit scans are complete. Finally, in step **425** the analyzer component **123** is initialized so that it can receive and store the results of audit scans such as a network element's functions and vulnerabilities. In alternative embodiments of the present invention, the functions performed by the analyzer component **123**, the decision maker component **121**, the blackout manager **124**, and the job manager component **122** can be performed by other components separate from the active scan engine **120**.

[**0049**] **FIG. 5**. illustrates an exemplary means for recovering prior data as referred to in step **310** of **FIG. 3**. The purpose of this step is to continue scanning, analysis, or decision-making work that was interrupted during a previous shut down of the security audit system **115**. In step **505**, analysis is completed of any remaining hosts in the network **110** that have not already been analyzed. In step **510**, the active scan engine **120** checks for a user-initiated shutdown of the security audit system **115**, or proceeds to step **515** to analyze data remaining from any unprocessed scan jobs. After the prior data has been recovered, the active scan engine **120** once again checks for a user-signaled shutdown in step **520**. If a shutdown has been initiated, the active scan engine **120** shuts down; otherwise, it returns to step **315** for adjusting or configuring new scans.

[**0050**] An exemplary method for configuring scans is illustrated in FIGS. **6**A and the accompanying exemplary tables. The first time the security audit system **115** runs on

a network **110**, a scan configuration table can be created after a discovery scan is performed. Subsequently, the active scan engine **120** stores the scan configuration table and, with each new scan, the table can be updated. Referring to **FIG. 6**A, in step **605** the active scan engine **120** determines if the decision maker **121** is due to be executed. The decision maker **121** runs periodically, or when a discovery scan finds previously unknown hosts on the network **110**. In step **610**, the decision maker **121** opens the scan configuration table in the ASE database **125**. As shown in an exemplary table in **FIG. 6**D, the scan configuration table contains the necessary information for configuring and scheduling scans of all known hosts. In step **615**, the decision maker **121** reads a host's scan configuration record and, in step **620**, the decision maker **121** examines the host's scan configuration record. In step **625**, if the host is due for an audit scan, the host is added to the set of hosts to be scanned with the scan policy indicated in the scan configuration record. The decision maker **121** checks for a user-initiated shutdown in step **630**, and proceeds to shut down if so indicated. Otherwise, in step **635** the decision maker **121** checks for more host scan configuration records. If there are more records to read, the process **315** returns to step **615**, and is repeated with the next record. If there are no records remaining to be processed, the decision maker **121** resets itself in step **640**. This step involves setting parameters governing the next periodic run of the decision maker **121**.

[**0051**] The type of scan that is run on each host for an element in the network **110** can be selected manually by the user or done automatically by the active scan engine **120**. The advantage of automating the scan configuration is that there are often numerous elements to scan in a network and hundreds of possible scanning checks that can be performed on each element. By automating the process, configuring and scheduling scans of each element of the network can be performed periodically, or whenever a new element is found during a discovery scan **FIG. 7** illustrates an exemplary decision step for determining whether to schedule a discovery scan or an audit scan. A discovery scan is conducted periodically, or when a network is being audited for the first time. If a discovery scan is being conducted in step **710**, the active scan engine **120** can follow the exemplary method for scheduling a job illustrated in **FIG. 8**. If an audit scan is going to be conducted in step **715**, the active scan engine **120** locates other hosts with the same scan policy as shown in **FIG. 9**.

[**0052**] A scan policy comprises a list of vulnerabilities to be checked during a scan. Scanning all the hosts with the same policy at one time allows the active scan engine **120** to coordinate scans efficiently. Once the other hosts with the same policy are located in step **905**, a job is scheduled for the audit scan in step **910** in the same way that a job is scheduled for a discovery scan. If more host ranges have been configured for scans, step **915** will return to step **905** and the process will repeat for the next host range. Otherwise, the process is complete.

[**0053**] An exemplary method for scheduling a job is illustrated in **FIG. 8**. In step **805**, a unique job identifier is assigned to the scan job. In step **810**, the active scan engine **120** estimates the duration of the scan and checks with the blackout manager, in step **815**, for a clear time period of at sufficient duration for scanning. Once a time period is selected for the scan job, in step **820** a start time is scheduled. In step **825**, the job is registered with the job manager **122** in the active scan engine **120** so that it can be tracked. In step **830**, if the scheduled job is for a discovery scan, the process is complete. Otherwise, in step **835** the last scan job identifier is updated for each host included in the newly scheduled job.

[**0054**] Once a job is scheduled, the scan is ready to run against the appropriate elements in the network **110**. An exemplary method for running a scan, as referred to in step **325** of **FIG. 3**, is illustrated in greater detail in **FIG. 10**. In step **1005**, the active scan engine **120** retrieves the next available job from the job manager. Depending on the type of scan that is to be run, in step **1010**, the active scan engine **120** starts the appropriate scan engine. In steps **1015** and **1020**, the active scan engine **120** sends a request to the appropriate scan engine to start the job and updates the job status to active. Running the scan collects data about one or more of the various elements in the network and returns that data to the active scan engine **120** for analysis.

[**0055**] Referring to **FIG. 11**, an exemplary method for analyzing scan results, as referenced in step **330** of **FIG. 3** and step **515** of **FIG. 5**, is illustrated. In step **1105**, the analyzer component **123** selects the next scan job to analyze. In step **1110**, the analyzer **123** selects the next host to process in the current scan job's data. If this is the first time that the host is examined, then the data are from a discovery scan. The current host's scan configuration record is written or updated in step **1115**. Based on the host function and asset value, further scan policies can also be automatically selected by the scan engine for future audit scans against the host. In steps **1120** and **1125**, the analyzer **123** processes the vulnerability and service records for the current host.

[**0056**] If the scan was a discovery scan, the analysis process ends at step **1130**. However, if the scan was an audit scan, the analysis process continues in step **1135**, where a new security score is computed for the host. An advantageous means for calculating a security score is described in U.S. non-provisional patent application entitled "Method and System for Calculating Risk Associated with a Security Audit," filed concurrently herewith, having attorney docket number 05456.105036. An exemplary method for processing a security score is discussed in greater detail with reference to **FIG. 19**. In step **1140**, the host's previous security score is updated. If the user initiated a shutdown of the security audit system **115** at this time, any unanalyzed host information or job information is saved in steps **1150** and **1155**. If there are more hosts to be analyzed or more scan jobs to be performed in steps **1165** and **1170**, the process will return to the beginning and repeat.

[**0057**] In **FIG. 12**, an exemplary method for populating a host's scan configuration, as referenced in step **1115** of **FIG. 11**, is illustrated. In step **1203**, the analyzer **123** examines the state data for the current host. These data indicate what operating system and services a host is running, and form the basis for computing the host's asset value in step **1205** and its role in step **1210**. In step **1215**, the analyzer **123** retrieves the scan policy to be applied to the host based on its asset value and role. **FIG. 6B** is an exemplary table that maps a host role and asset value to a scan policy. In step **1218**, the scan frequency of the host is computed as a function of its asset value. **FIG. 6C** illustrates an exemplary table for mapping a host's asset value to a scan frequency. Both of the

mapping tables illustrated in **FIGS. 6B and 6C** may be adjusted and customized by the user. The analyzer **123** writes or updates the host's scan configuration record in step **1223**.

[**0058**] **FIG. 13** illustrates an exemplary method for computing a host's asset value, as referenced in step **1205** of **FIG. 12**. In step **1305**, the analyzer **123** retrieves from the ASE database **125** the asset value associated with the host's operating system. Asset values assigned on the basis of operating system reflect the greater importance of servers and routers than regular desktop systems. In step **1310**, the analyzer **123** retrieves the asset value associated with the host's running services. Asset values are associated with a number of services in the ASE database **125**, and reflect the relative importance of the various services. In step **1310**, the analyzer **123** selects the maximum asset value associated with any of the services that are active on the host. In step **1315**, the analyzer **123** retrieves the asset value associated with the host's vulnerabilities. As with services, the presence of some vulnerabilities can indicate a greater importance for a host; the asset values associated with vulnerabilities in the ASE database **125** reflect this. In the present example, the maximum asset value associated with any of the host's vulnerabilities is chosen. In step **1320**, the analyzer **123** selects the maximum of the previously retrieved asset values as the host's asset value.

[**0059**] **FIG. 14** illustrates an exemplary method for computing a host's role, as referenced in step **1210** of **FIG. 12**. Steps **1405**, **1425**, and **1445** distinguish between various exemplary host operating systems. The next layer of decisions (steps **1410** and **1430**) distinguish between web servers and non-web servers. As shown in the figure, the exemplary host roles identified are NTDesktop (step **1415**), NTWebServer (step **1420**), UnixWebServer (step **1435**), UnixDesktop (step **1440**), Router (step **1450**), and Unknown (step **1455**). While **FIG. 14** shows an exemplary implementation of determining a host's role on a network, those skilled in the art will recognize that other host parameters may be taken into account, yielding a greater variety of roles.

[**0060**] Referring to **FIG. 15**, an exemplary method for processing vulnerability records is illustrated. In step **1505**, the analyzer **123** queries the vulnerability history table for the current host. In step **1510**, a vulnerability identified during the audit scan is located and the vulnerability history table is examined for the same listing in step **1515**. If the vulnerability is listed in the history table, the state table and history table are updated for the particular host in step **1520**. As shown in greater detail in the exemplary logic flow diagram in **FIG. 16**, step **1520** involves the analyzer **123** updating the record in the state table and the record in the history table. If the vulnerability is not found in the history table in step **1515**, the analyzer **123** writes a new record to the state table and history table in steps **1530** and **1535**. If there are more vulnerabilities to be examined, the process returns to step **1510**. Once the vulnerability records for a host are processed, the logic flow diagram returns to step **1125** for processing service records.

[**0061**] An exemplary method for processing service records, as referred to in step **1125** of **FIG. 11**, is illustrated in **FIG. 17**. In step **1705**, the analyzer **123** queries the service history table for the current host that is being examined. In step **1710**, a service identified during the audit scan is located and the service history table is examined for the same listing in step **1715**. If the service is listed in the history table, the analyzer **123** updates the state table and history table for the particular host in step **1720**. As shown in greater detail in the exemplary logic flow diagram in **FIG. 18**, step **1720** involves updating the record in the state table and the record in the history table. If the service is not found in the history table in step **1715**, the analyzer **123** writes a new record to the state table and history table in steps **1730** and **1735**. If there are more services to be examined, the process returns to step **1710**. Once the service records for a host are processed, the logic flow diagram returns to step **1130**.

[**0062**] An exemplary method for processing the security score of a host is illustrated in **FIG. 19**. In step **1905**, the active scan engine **120** queries the vulnerability state table for the set of vulnerabilities for the current host. The vulnerabilities were previously detected by the various scans performed on the particular host. The active scan engine **120** selects the next vulnerability in the state table in step **1910** and calculates a risk for the vulnerability in step **1915**. An exemplary method for calculating a risk in association with a security audit of a computer network is taught in the related application referenced herein. In step **1920**, the vulnerability count for the risk band that includes the calculated risk is incremented. If there are additional vulnerabilities in the state table, the process is repeated and the vulnerability count for the appropriate band is incremented. When there are no remaining active vulnerabilities for this host in the state table, a logarithmic band calculation is applied to the accumulated risks in step **1930**. An exemplary method for performing a logarithmic band calculation on accumulated risks is taught in the related application referenced herein.

[**0063**] The security audit system **115** keeps a record of security scores over time. In step **2005** of **FIG. 20**, the most current host security score is retrieved from the host table. If the most current security score is different from the newly calculated score in step **2010**, the active scan engine **120** writes a new record to the host history table in step **2015**, and updates the host's current security score in the host table in step **2020**. If the security score is not different in step **2010**, the process returns to step **1145** in **FIG. 11**. The forgoing steps illustrate an exemplary method for processing the security score of the host. In alternative embodiments of the invention other methods can be used to compute security scores for various elements in a distributed computing network.

[**0064**] **FIG. 21** illustrates an exemplary method for shutting down the active scan engine **120**, as referenced in various figures. The shutdown procedure consists of saving any state information necessary to resume any operations in progress at the time of shutdown. This information consists of information about jobs scheduled but not completed, jobs completed but not analyzed, and jobs incompletely analyzed.

[**0065**] In conclusion, the present invention enables and supports security auditing of a distributed computing network. The security audit system can conduct a discovery scan of the network to identify network elements and determine their function, vulnerabilities, and relative importance. Using this information, more comprehensive audit scans are scheduled to regularly assess and monitor the

security of the network. The security audit system can automatically select particular audit scans based on the types of hosts identified in the network. The audit scans can be automatically scheduled so as not to interfere with the regular functions of the network. Information collected during the audit scans can also be used to compute a security score for a network element.

[0066] It will be appreciated that the present invention fulfills the needs of the prior art described herein and meets the above-stated objects. While there has been shown and described the preferred embodiment of the invention, it will be evident to those skilled in the art that various modifications and changes may be made thereto without departing from the spirit and the scope of the invention as set forth in the appended claims and equivalence thereof. Although the present invention has been described as operating on a local area network, it should be understood that the invention can be applied to other types of distributed computing environments. Furthermore, it should be readily apparent that the components of the security audit system can be located in various local and remote locations of a distributed computing environment.

What is claimed is:

1. A computer-implemented method for configuring and scheduling a security audit of a computer network comprising the steps of:

conducting a discovery scan to identify an element of the computer network and determine the element's functions;

configuring an audit scan to perform on the element, wherein the audit scan is a more thorough scan than the discovery scan;

scheduling a time to perform the audit scan on the element;

running the audit scan of the element at the scheduled time;

calculating a security score for the element based on the audit scan; and

scheduling another time to repeat the audit scan on the element, the scheduling based on the results of the audit scan.

2. The method of claim 1, further comprising the step of configuring a subsequent audit scan of the element that is different from the audit scan.

3. The method of claim 1, further comprising the step of receiving a blackout time during which no audit scan can be scheduled.

4. The method of claim 1, wherein the step of conducting a discovery scan further comprises identifying the function of the element.

5. The method of claim 1, wherein the step of conducting a discovery scan further comprises identifying vulnerabilities associated with the element.

6. The method of claim 1, wherein the step of conducting a discovery scan further comprises assigning an asset value for the element, wherein the asset value indicates the relative importance of the element in the network.

7. The method of claim 6, wherein the asset value is modified based on the audit scan.

8. The method of claim 1, further comprising the step of receiving a manually selected asset value for the element.

9. The method of claim 1, wherein the step of configuring an audit scan comprises selecting a type of audit scan based on the discovery scan.

10. The method of claim 1, wherein the step of configuring an audit scan comprises:

retrieving an asset value based on the discovery scan;

retrieving a scan frequency associated with the asset value, wherein the scan frequency indicates how often the scan is performed; and

assigning a role based on the discovery scan, wherein the role indicates the function of the element; and

assigning a policy based on the discovery scan, wherein the policy indicates the type of audit scan.

11. The method of claim 1, wherein the step of configuring an audit scan comprises manually selecting the type of audit scan.

12. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 1.

13. A computer-implemented method for configuring and scheduling a security audit of a computer network comprising the steps of:

conducting a discovery scan to identify an element of the computer network;

configuring an audit scan to perform on the element;

scheduling a time to perform the audit scan on the element; and

running the audit scan at the scheduled time on the element.

14. The method of claim 13, further comprising the step of calculating a security score for the element based on the audit scan.

15. The method of claim 13, further comprising the step of scheduling another time to perform the audit scan on the element.

16. The method of claim 13, further comprising the step of receiving a blackout time during which no audit scan can be scheduled.

17. The method of claim 13, wherein the step of conducting a discovery scan further comprises identifying at least one of the functions or vulnerabilities associated with the element.

18. The method of claim 13, wherein the step of conducting a discovery scan further comprises assigning an asset value for the element.

19. The method of claim 13, wherein the step of configuring an audit scan comprises:

retrieving an asset value based on the discovery scan;

retrieving a scan frequency associated with the asset value; and

assigning a role and a policy based on the discovery scan.

20. The method of claim 13, wherein the step of configuring an audit scan comprises manually selecting the type of audit scan.

21. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 1.

22. A method for assessing the security of a network comprising the steps of:

receiving an initial scan identifying a network element and the function of the network element;

selecting an audit scan to perform on the network element, the selection based on the initial scan, wherein the audit scan is more thorough than the initial scan;

scheduling the audit scan to perform on the network element;

performing the audit scan on the network element at the scheduled time;

receiving data from the selected audit scan of the network element; and

computing a security score for the network element from the selected audit scan.

23. The method of claim 22, further comprising modifying the selected audit scan; said modification based on the data received from the selected audit scan.

24. The method of claim 22, wherein the step of receiving an initial scan comprises:

identifying an operating system for the network element;

identifying a service for the network element, the service indicating the element's function;

determining an asset value of the network element from the operating system and the service of the network element, the asset value indicating the relative importance of the network element; and

identifying a vulnerability associated with the network element.

25. The method of claim 22, wherein the step of selecting an audit scan is based on the initial scan.

26. The method of claim 22, wherein the step of selecting an audit scan is based on a manual input.

27. The method of claim 22, wherein the step of scheduling the audit scan comprises checking a blackout schedule.

28. The method of claim 22, wherein the step of computing a security score comprises summing one or more vulnerabilities associated with the element.

29. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 22.

30. A method for assessing the security of a network comprising the steps of:

receiving an initial scan identifying a network element;

selecting an audit scan to perform on the network element, said selection based on the initial scan;

performing the selected audit scan on the network;

receiving data from the selected audit scan of the network element; and

computing a security score for the network element from the selected audit scan.

31. The method of claim 30, further comprising the step of scheduling the selected audit scan, said scheduling based on the initial scan.

32. The method of claim 30, further comprising modifying the selected audit scan, said modification based on the data received from the selected audit scan.

33. The method of claim 30, wherein the step of receiving an initial scan comprises:

identifying an operating system and a service for the network element;

determining an asset value of the network element from the operating system and the service of the network element; and

identifying a vulnerability associated with the network element.

34. The method of claim 30, wherein the step of selecting an audit scan is based on the initial scan.

35. The method of claim 30, wherein the step of selecting an audit scan is based on a manual input.

36. The method of claim 30, wherein the step of scheduling the audit scan comprises checking a blackout schedule.

37. The method of claim 30, wherein the step of computing a security score comprises summing one or more vulnerabilities associated with the element.

38. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 30.

39. A system for configuring and scheduling a security audit of a computer network comprising:

the computer network;

a security audit system operable for conducting a discovery scan to identify an element of the computer network and configuring and scheduling an audit scan of the element; and

a console operable for receiving information from the security audit system and transmitting information to the security audit system about the discovery scan and the audit scan.

40. The system of claim 39, wherein the security audit system is further operable for conducting a discovery scan to:

identify a function for the element;

determine an asset value for the element; and

identify a vulnerability for the element.

41. The system of claim 39, wherein the security audit system checks a blackout schedule before scheduling an audit scan.

42. The system of claim 39, wherein the security audit system further comprises a system scanning engine operable for detecting particular vulnerabilities on the network element.

43. The system of claim 39, wherein the security audit system further comprises an Internet scanning engine operable for performing a discovery scan on the network.

44. The system of claim 39, wherein the security audit system further comprises a database scanning engine operable for detecting vulnerabilities in database elements within the network.

45. The system of claim 39, wherein the security audit system further comprises an active scan engine operable for selecting, coordinating, and scheduling various discovery and audit scans to be performed on the computer network.

* * * * *

Exhibit A-7

US 20020026591A1

(19) United States

(12) Patent Application Publication

Hartley et al.

(10) Pub. No.: US 2002/0026591 A1

(43) Pub. Date: Feb. 28, 2002

(54) METHOD AND APPARATUS FOR ASSESSING THE SECURITY OF A COMPUTER SYSTEM

(76) Inventors: **Bruce V. Hartley**, Elbert, CO (US); **Eric Knight**, Pueblo West, CO (US); **Greg Zymbaluk**, Colorado Springs, CO (US); **Cynthia Mavros**, Palmer Lake, CO (US); **Kevin Reynolds**, Pueblo, CO (US)

Correspondence Address:
Gordon R. Lindeen III
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
Seventh Floor
12400 Wilshire Boulevard
Los Angeles, CA 90025-1026 (US)

(21) Appl. No.: **09/834,334**

(22) Filed: **Apr. 12, 2001**

Related U.S. Application Data

(63) Continuation of application No. 09/333,547, filed on Jun. 15, 1999, now abandoned, which is a non-provisional of provisional application No. 60/091,270, filed on Jun. 15, 1998.

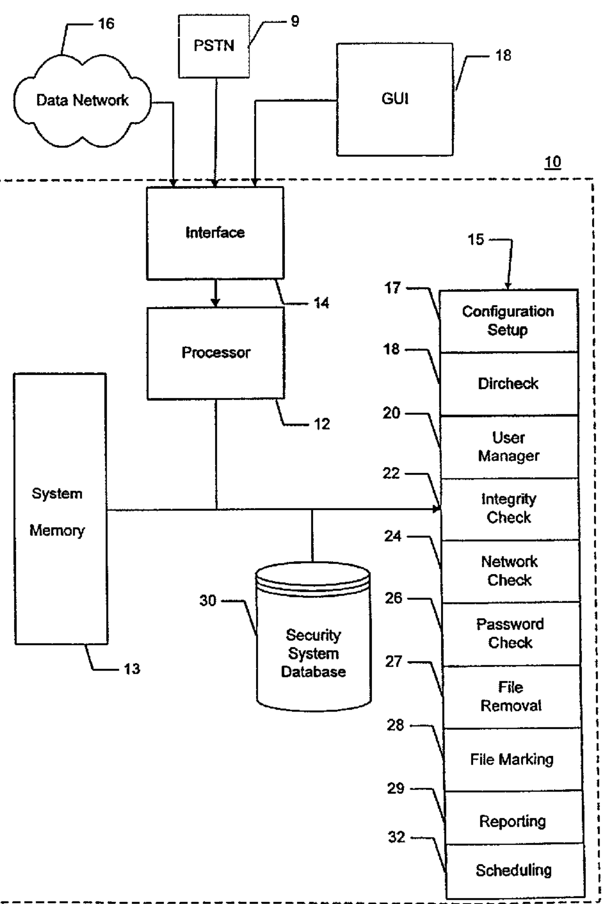
Publication Classification

(51) Int. Cl.[7] **G06F 11/30**; H04L 9/00

(52) U.S. Cl. **713/201**; 709/224

(57) **ABSTRACT**

A method and apparatus performs a security analysis computer system to identify, notify, and possibly correct, vulnerabilities and discrepancies. The security system includes a number of security tools and utilities in order to perform these functions. The security system includes the capability to identify the system configuration and once this is done performs different processes to analyze the computer system directories, locate vulnerabilities in the files or directories, check the network access, do analysis of the users or groups which have access to the computer system and check the permissions which these parties have been granted, and analyze passwords of the users. The utilities include the functionality to permanently remove files from the computer system, mark particular files to be analyzed, as well as schedule the security tests to be performed at predetermined times.



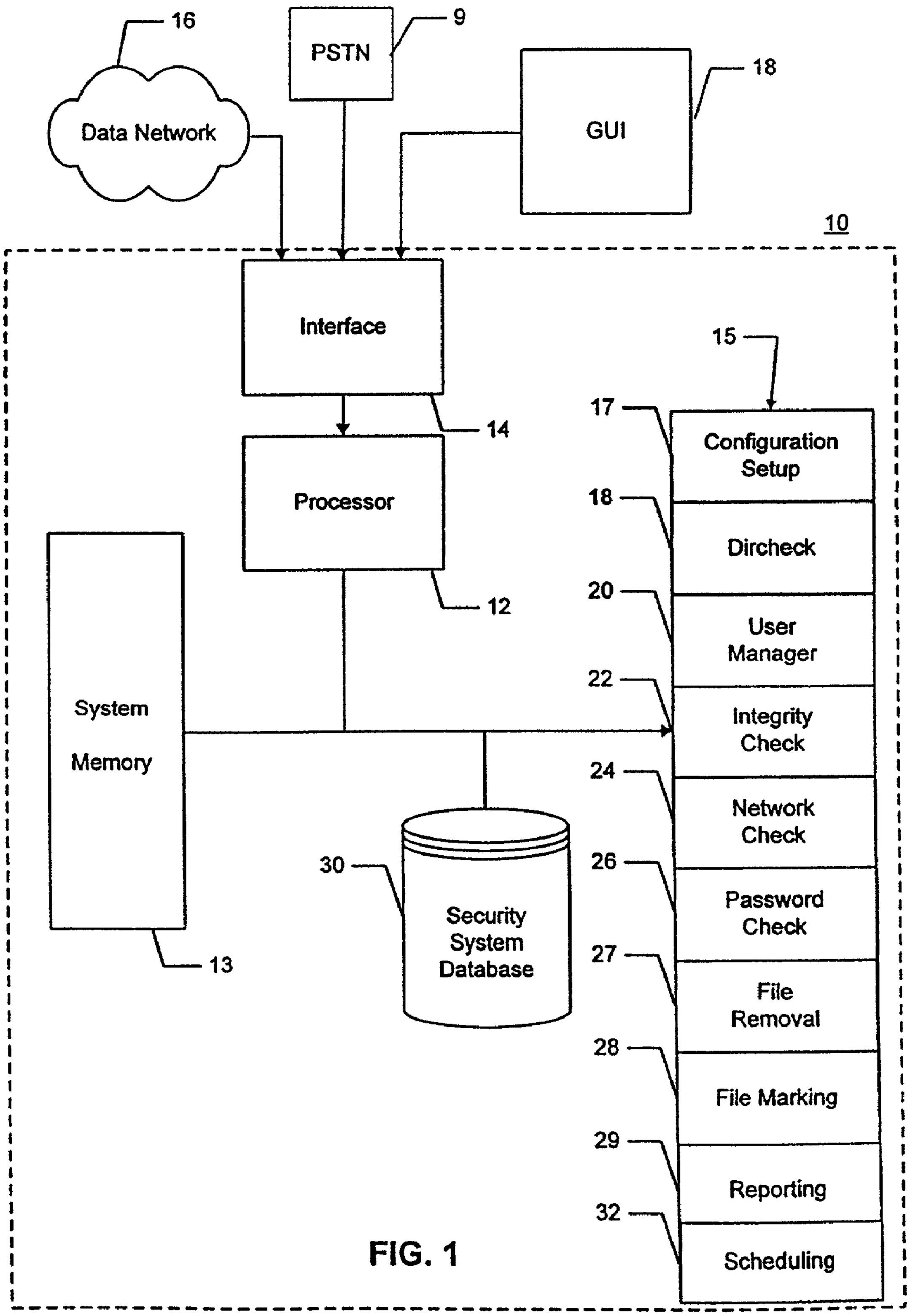

16
Data Network
PSTN
9
GUI
18
10
Interface
15
17
14
Configuration Setup
Processor
18
Dircheck
20
12
User Manager
22
System
Memory
Integrity Check
24
Network Check
30
26
Security System Database
Password Check
13
27
File Removal
28
File Marking
29
Reporting
32
Scheduling

FIG. 1

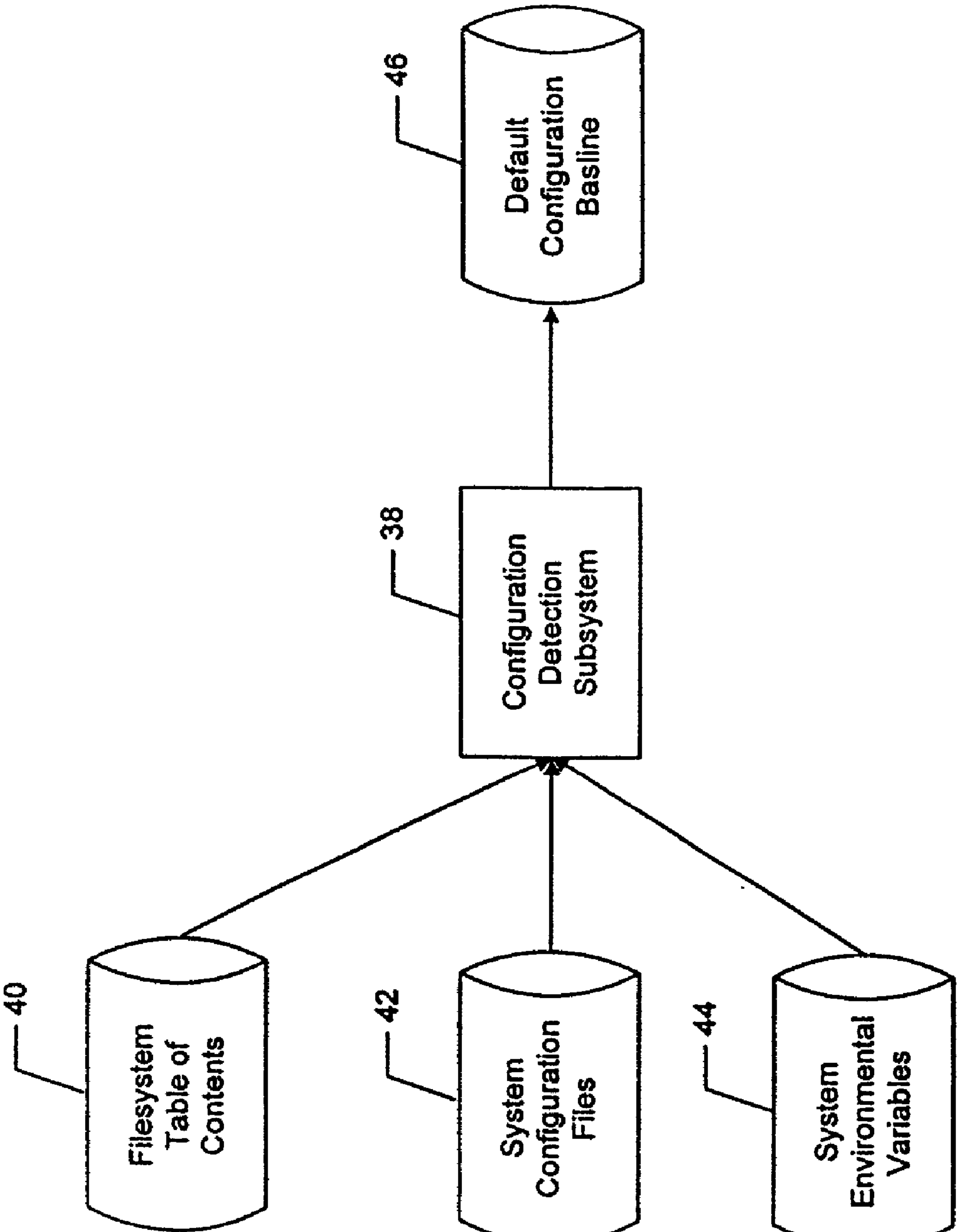

46
Default Configuration Basline
38
Configuration Detection Subsystem
40
Filesystem Table of Contents
42
System Configuration Files
44
System Environmental Variables

Fig. 2

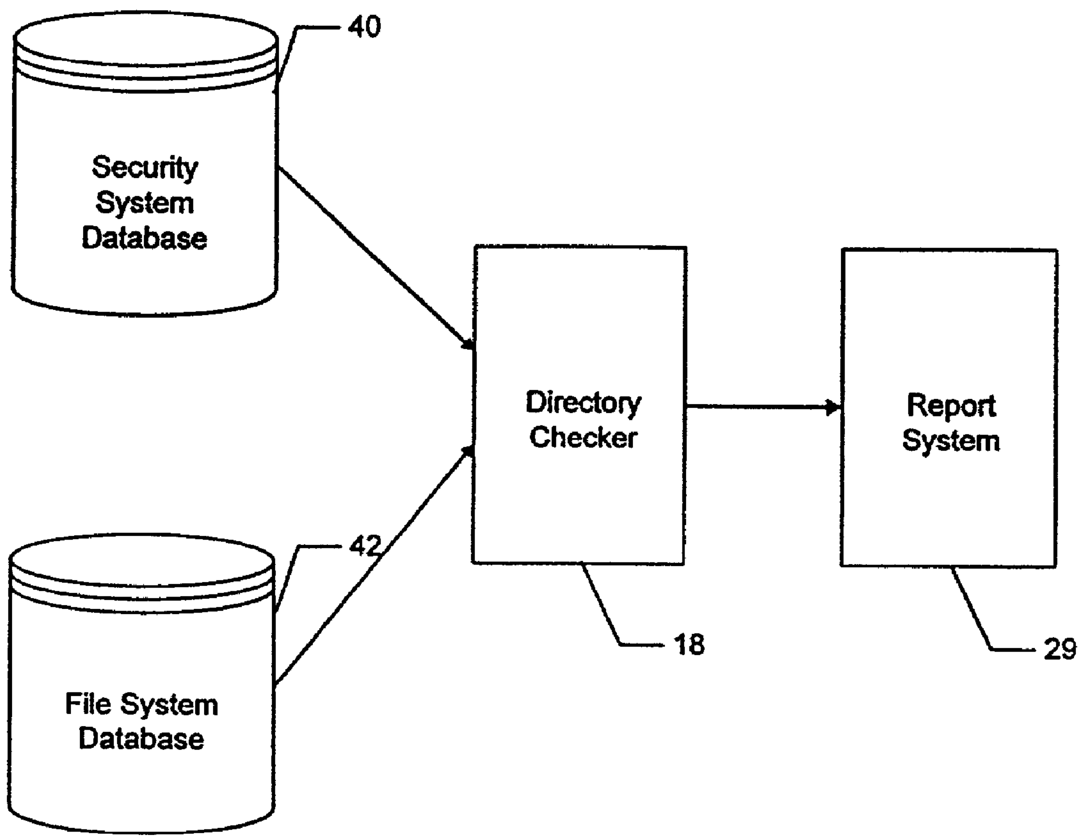

40
Security System Database
42
File System Database
Directory Checker
18
Report System
29

FIG. 3

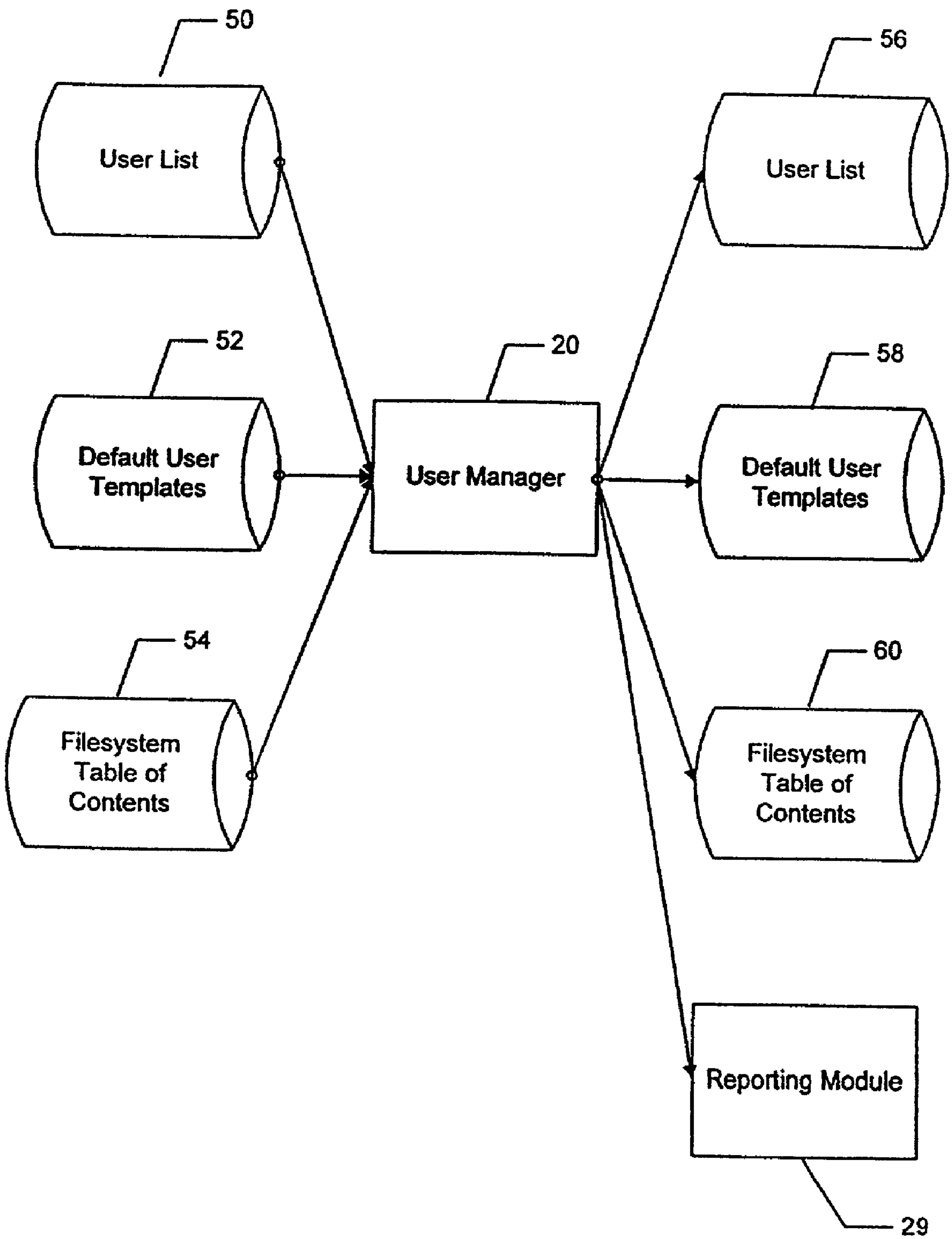

50
User List
52
Default User Templates
54
Filesystem Table of Contents
20
User Manager
56
User List
58
Default User Templates
60
Filesystem Table of Contents
Reporting Module
29

FIG. 4

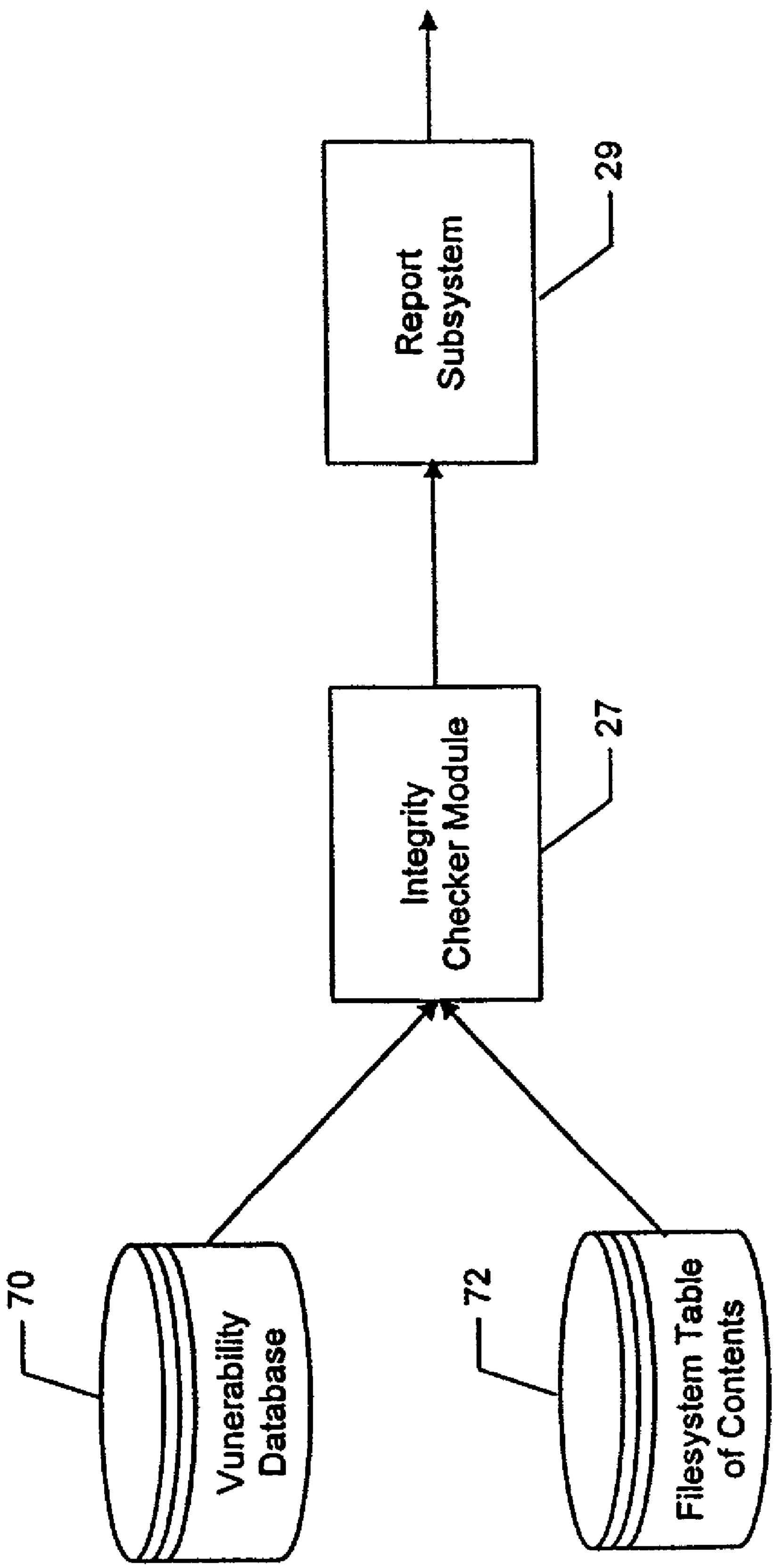

70
Vunerability Database
72
Filesystem Table of Contents
Integrity Checker Module
27
Report Subsystem
29

Fig. 5

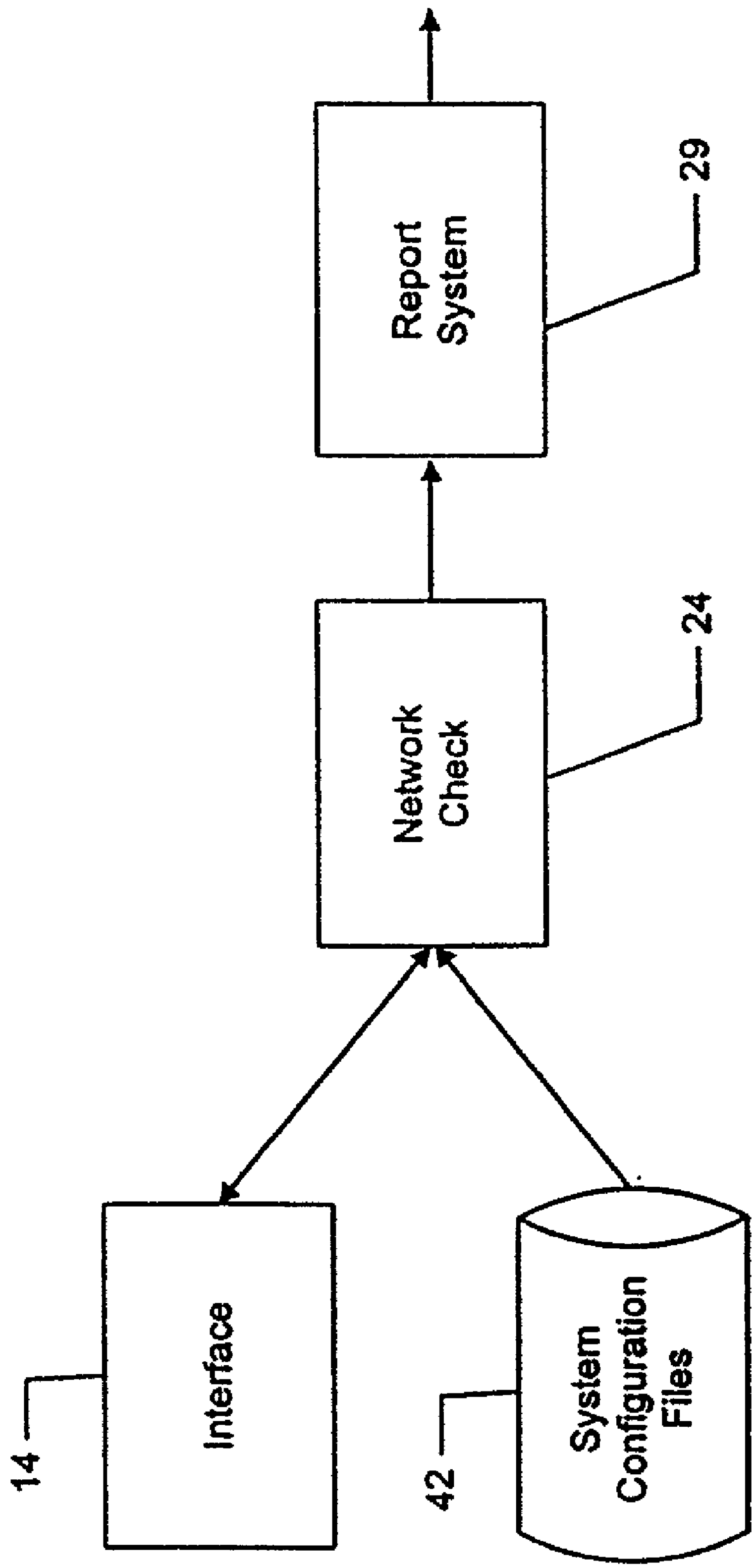

Report System
29
Network Check
24
Interface
14
System Configuration Files
42

Fig. 6

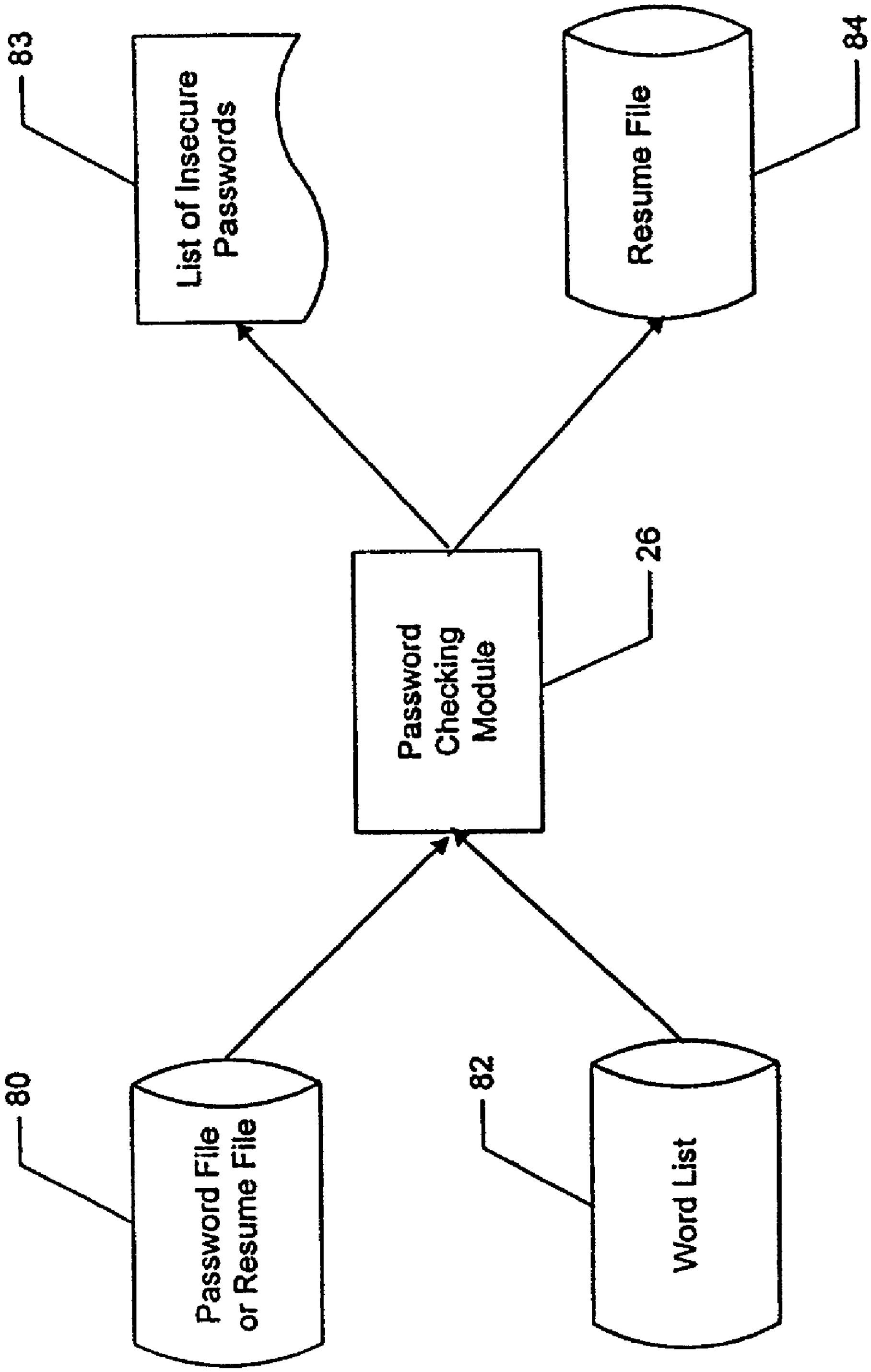

83
List of Insecure Passwords
Resume File
84
Password Checking Module
26
80
Password File or Resume File
82
Word List

Fig. 7

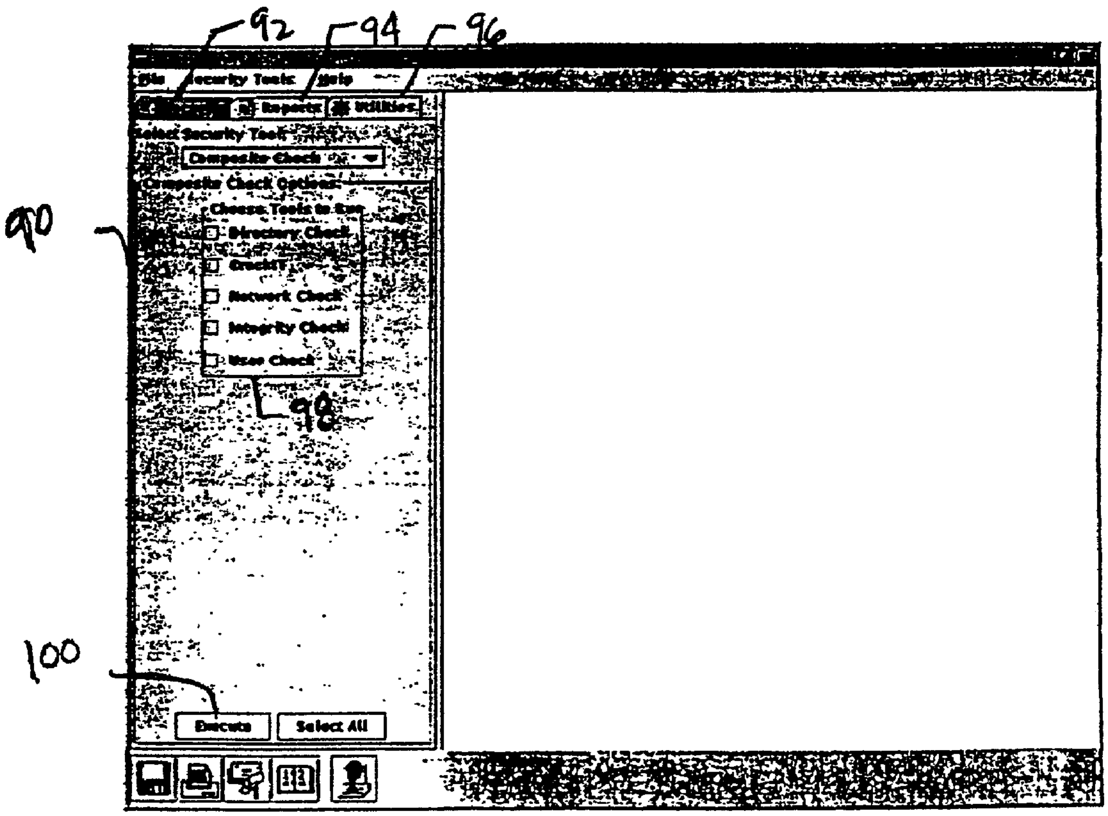

92
94
96
90
File
Security Tools
Help
Reports
Utilities
Select Security Tool
Composite Check
Composite Check Options
Choose Tools to Run
Directory Check
Network Check
Integrity Check
User Check
98
100
Execute
Select All

FIG. 8

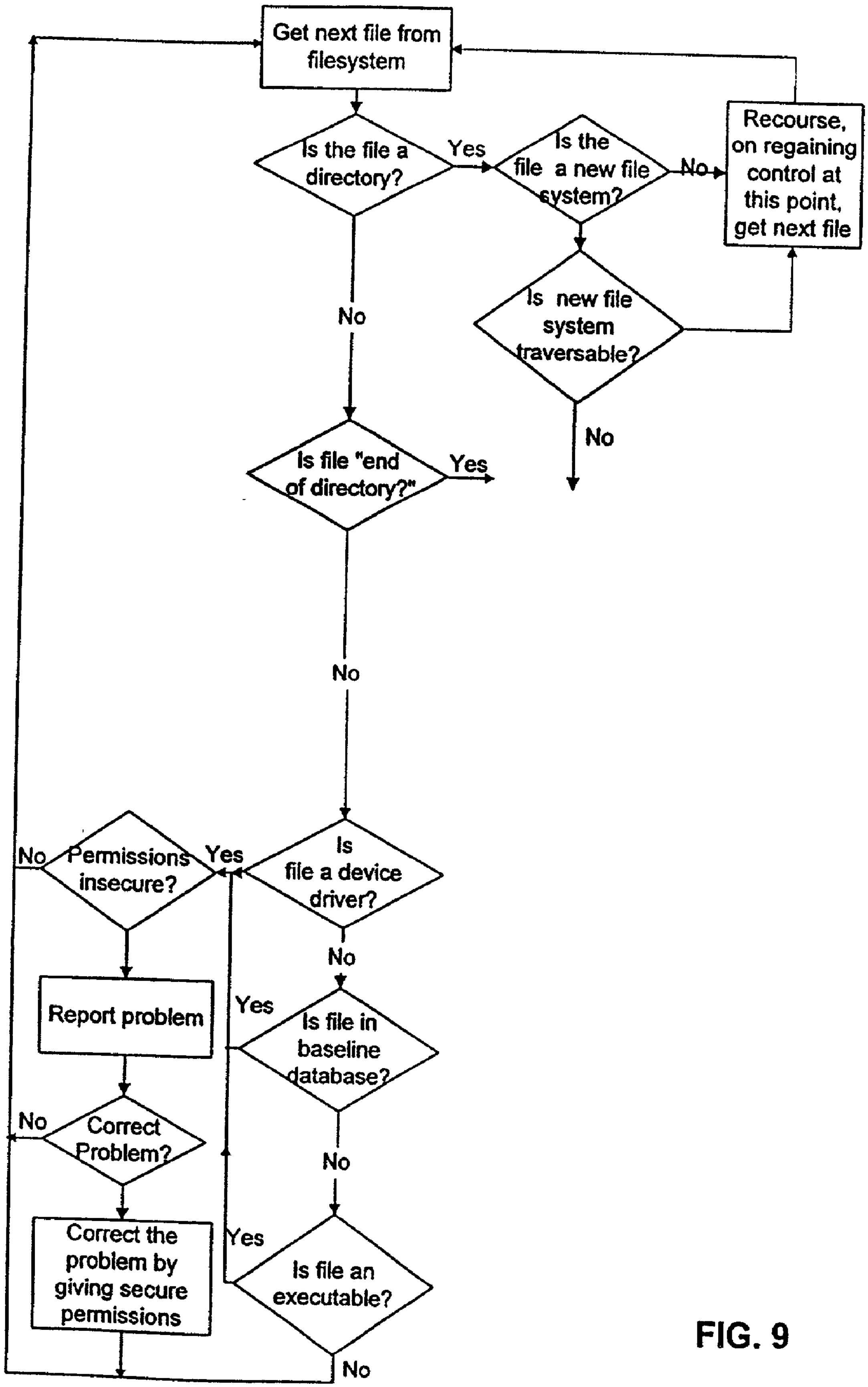

Get next file from filesystem
Is the file a directory?
Yes
Is the file a new file system?
No
Recourse, on regaining control at this point, get next file
Is new file system traversable?
No
No
Is file "end of directory?"
Yes
No
Is file a device driver?
Yes
Permissions insecure?
No
No
Report problem
Yes
Is file in baseline database?
No
Correct Problem?
No
Correct the problem by giving secure permissions
Yes
Is file an executable?
No

FIG. 9

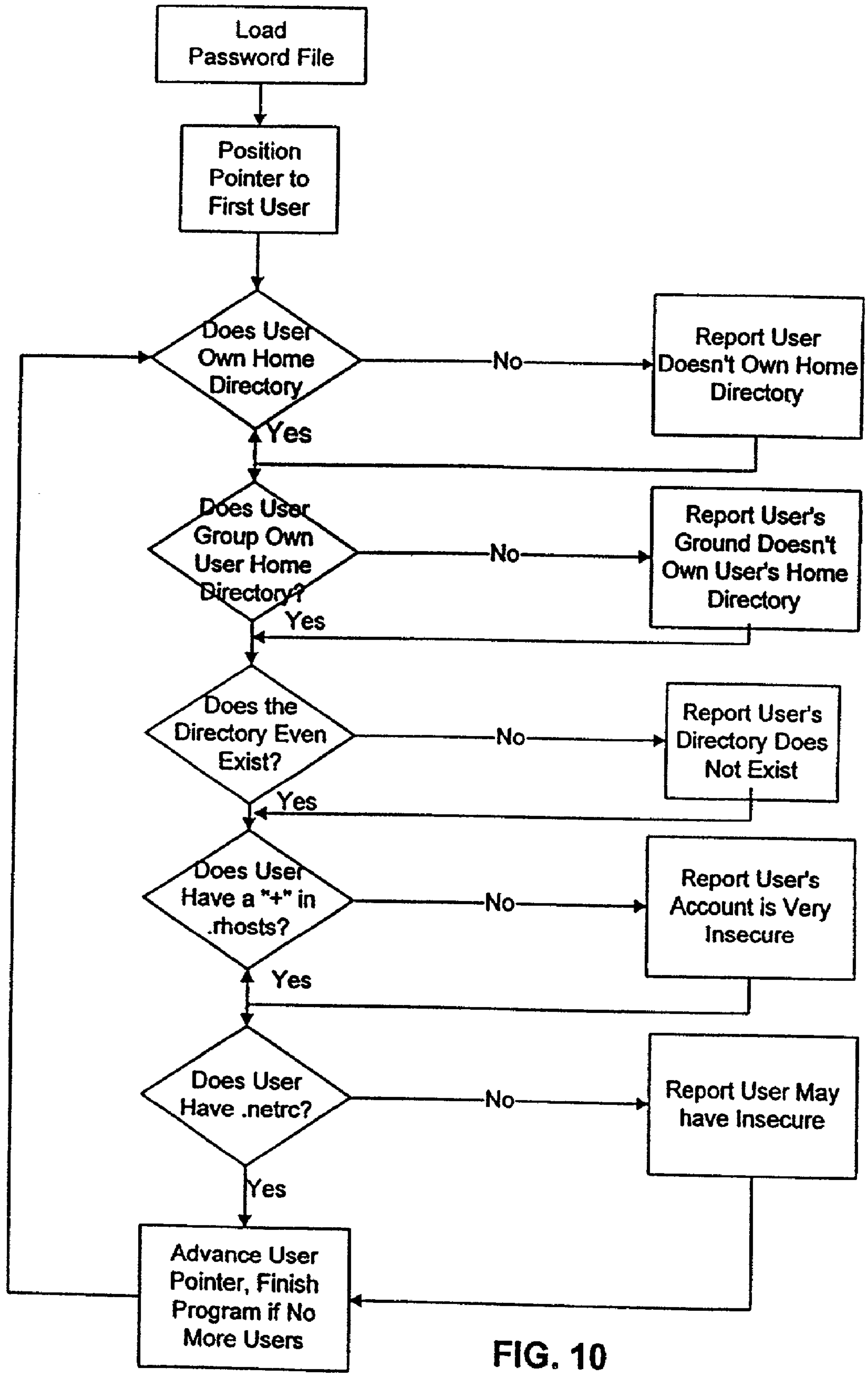

Load Password File
Position Pointer to First User
Does User Own Home Directory
No
Report User Doesn't Own Home Directory
Yes
Does User Group Own User Home Directory?
No
Report User's Ground Doesn't Own User's Home Directory
Yes
Does the Directory Even Exist?
No
Report User's Directory Does Not Exist
Yes
Does User Have a "+" in .rhosts?
No
Report User's Account is Very Insecure
Yes
Does User Have .netrc?
No
Report User May have Insecure
Yes
Advance User Pointer, Finish Program if No More Users

FIG. 10

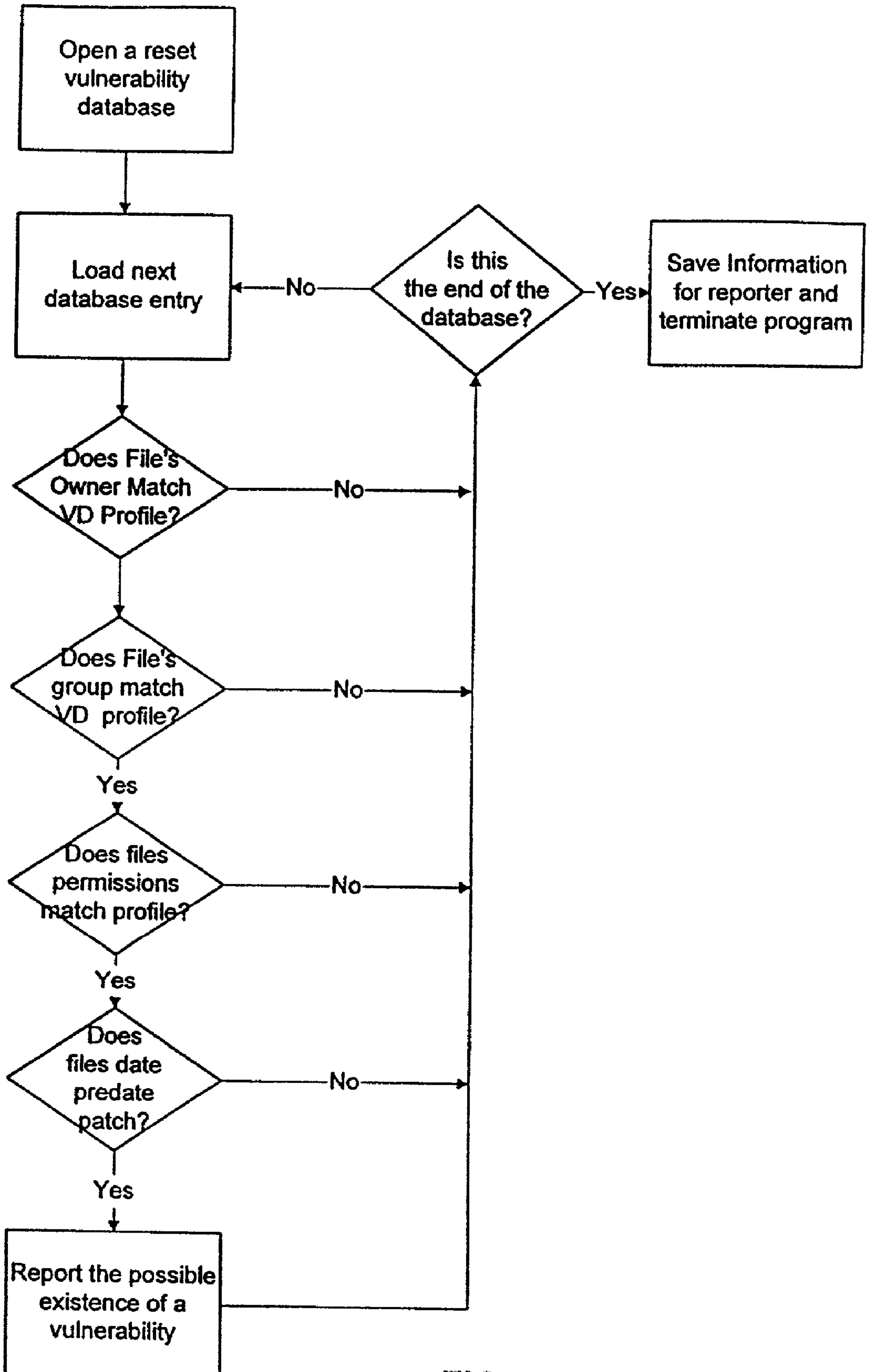

Open a reset vulnerability database
Load next database entry
No
Is this the end of the database?
Yes
Save Information for reporter and terminate program
Does File's Owner Match VD Profile?
No
Does File's group match VD profile?
No
Yes
Does files permissions match profile?
No
Yes
Does files date predate patch?
No
Yes
Report the possible existence of a vulnerability

FIG. 11

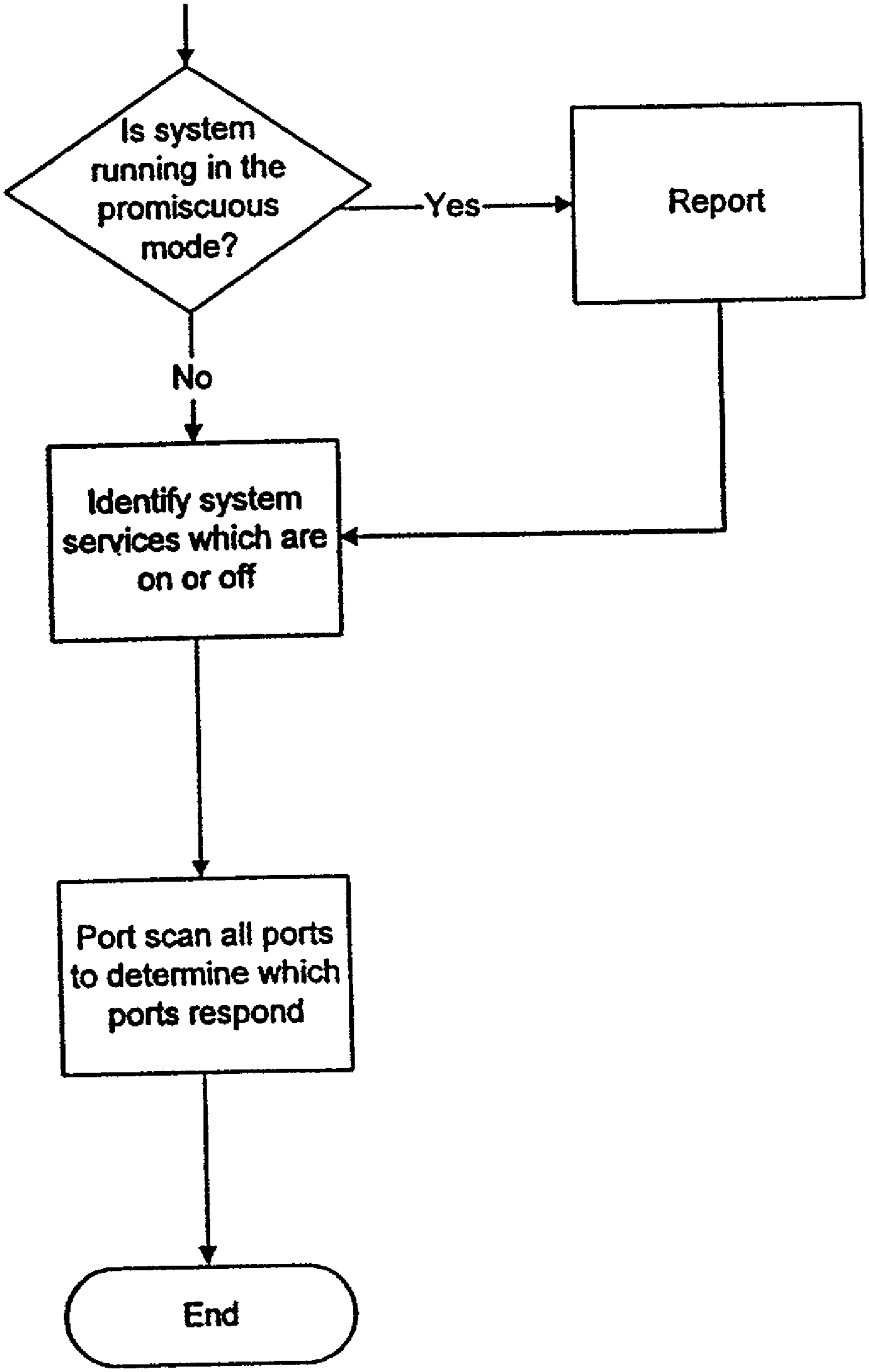

Is system running in the promiscuous mode?
Yes
Report
No
Identify system services which are on or off
Port scan all ports to determine which ports respond
End

Fig. 12

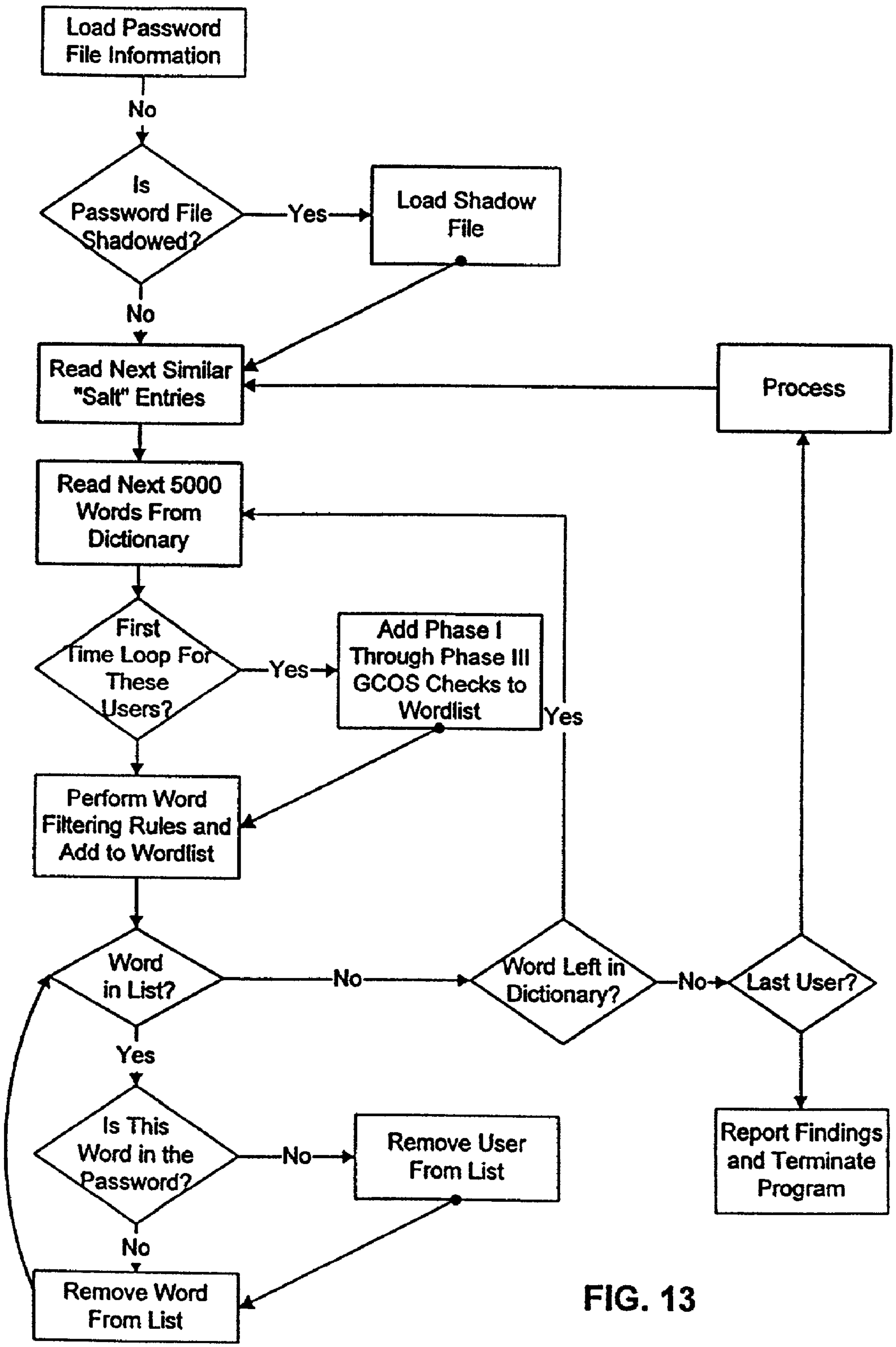

Load Password File Information
No
Is Password File Shadowed?
Yes
Load Shadow File
No
Read Next Similar "Salt" Entries
Process
Read Next 5000 Words From Dictionary
First Time Loop For These Users?
Yes
Add Phase I Through Phase III GCOS Checks to Wordlist
Yes
Perform Word Filtering Rules and Add to Wordlist
Word in List?
No
Word Left in Dictionary?
No
Last User?
Yes
Is This Word in the Password?
No
Remove User From List
Report Findings and Terminate Program
No
Remove Word From List

FIG. 13

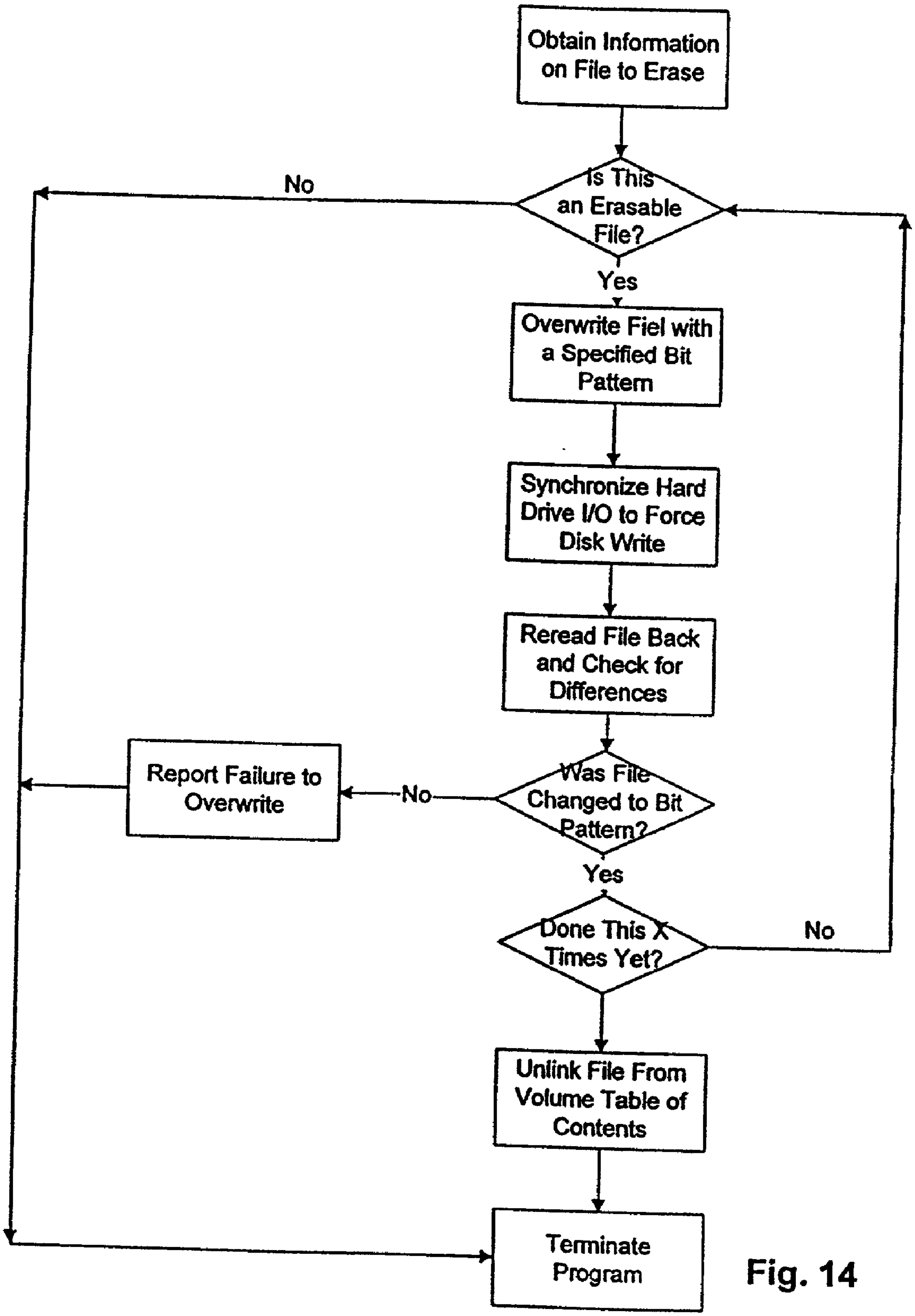

Obtain Information on File to Erase
Is This an Erasable File?
No
Yes
Overwrite Fiel with a Specified Bit Pattern
Synchronize Hard Drive I/O to Force Disk Write
Reread File Back and Check for Differences
Was File Changed to Bit Pattern?
No
Report Failure to Overwrite
Yes
Done This X Times Yet?
No
Unlink File From Volume Table of Contents
Terminate Program

Fig. 14

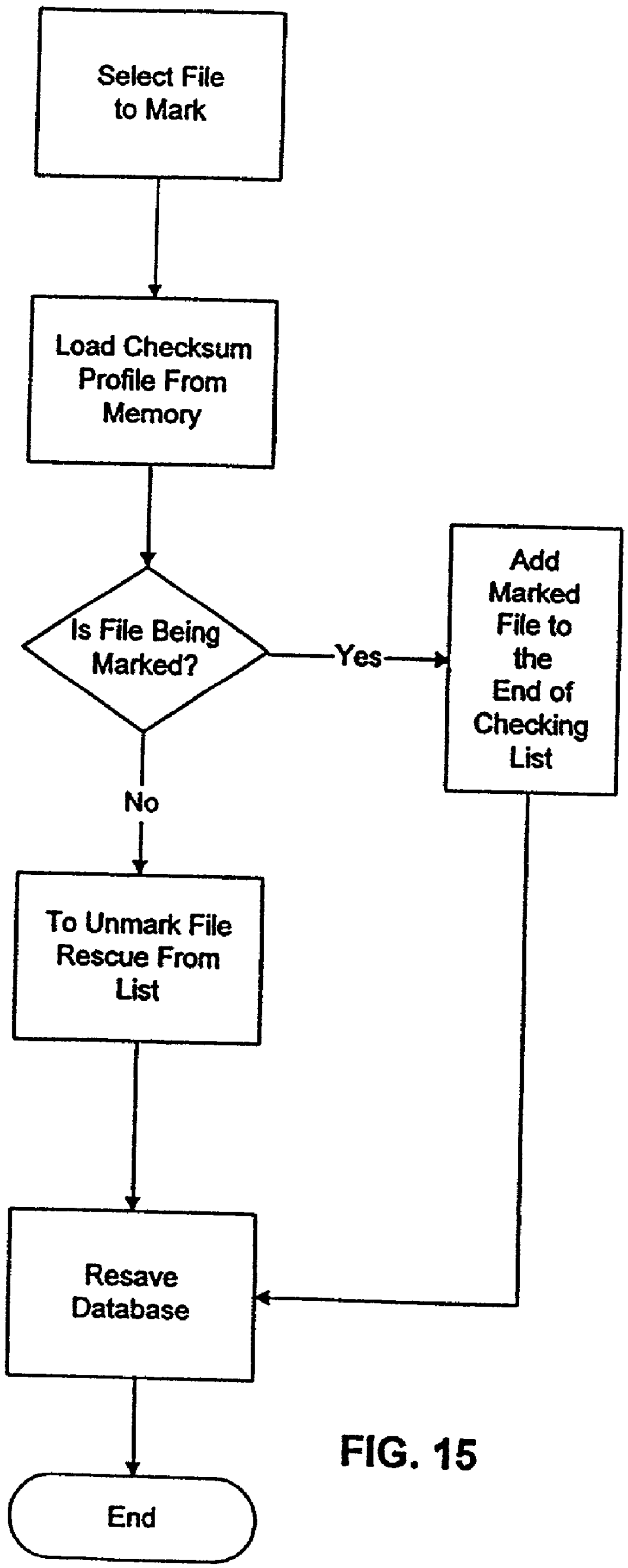

Select File to Mark
Load Checksum Profile From Memory
Is File Being Marked?
Yes
Add Marked File to the End of Checking List
No
To Unmark File Rescue From List
Resave Database
End

FIG. 15

METHOD AND APPARATUS FOR ASSESSING THE SECURITY OF A COMPUTER SYSTEM

FIELD OF THE INVENTION

[0001] The invention described herein relates to a method and apparatus for analyzing a computer system and identifying security vulnerabilities, and more specifically to a method and apparatus for performing a series of procedures which identify security vulnerabilities and discrepancies in the computer system and in some cases suggesting and implementing corrective action.

BACKGROUND OF THE INVENTION

[0002] As the use of computers has grown over the years, especially in business, there has been a growing need to develop computer systems which allow a number of individual computer users to communicate via their computers, and have access to common repositories of data. One solution had been to have all users within an organization connect to a single large main frame computer employing terminals with minimal processing capabilities. Another solution has been the development of server technology which allows a number of individual computer to connect to a central computer, i.e. server, which includes operating systems for a number of core functions for the network such as e-mail, common data bases, as well as a number of functions which are commonly employed by these computers connected to the network.

[0003] One advantage of employing server technology is that connections may be established to the server through a number of different modes. A first mode is a direct connection, such as through a local area network (LAN) The second type of connection may be made via a phone line from a remotely located computer. A connection may be established using the public switch telephone network (PSTN) with the server especially adapted to provide a telephonic connection. A third mode is a connection established to the server made over the Internet. With a connection established in this manner, system users browsing the web may access information stored on the server.

[0004] With these different modes to establish connections, it may be important to protect the information stored on a server from unauthorized access. Certain protections already exist such as requiring passwords when logging onto the server and restricting access to particular types of information only to designated parties.

SUMMARY OF THE INVENTION

[0005] The inventors have recognized that although many computer systems today include certain safeguards, such as passwords, for restricting access to the server and information contained therein, it is possible that these protections may be overcome. The inventors have further recognized that security vulnerabilities in a computer system may be identified and certain procedures may be performed within the computer system to reduce these vulnerabilities.

[0006] Described herein is a security system which identifies security vulnerabilities and discrepancies for a computer system. In some cases the security system may suggest corrections or provide fixes for the identified vulnerabilities and discrepancies. The computer system on which the security system resides may include a processor and an operational memory which contains all data which is to be analyzed by the security system described herein. The processor may direct a number of processing modules in the security system which perform various operations with regards to analyzing the computer system. The security system may also include a database which contains portions of data which may be employed by the processing modules in order to perform the various analysis of the computer system.

[0007] In one aspect of the invention, the security system includes at least one security module which analyzes files and directories resident in the system memory. The system may further include at least one utility module which may be employed to alert a system user to detected vulnerabilities, and provide corrective suggestions, and then implement the corrections when so directed. Included as part of the security modules may be a configuration detection device which analyzes the system to determine a configuration and located any unusual features. Once the configuration of the computer system has been determined, a directory check module function may be employed which detects security flaws that may have developed in the file system of the computer and determines if any, "security critical" files have been tampered with. A password security module may examine the passwords of the users with access to the computer system to detect insecure password choices. A network check module performs a number of processes to determine the vulnerability of the computer system when access may be gained via a data network.

[0008] Another security module may perform an integrity check which searches files in the computer system's operational memory and makes comparison against a store of known vulnerabilities. A user manager module performs an analysis of user accounts with regards to files and directories found in the operational memory. The user check may identify improper or invalid permissions and ownerships, associated with file analyzed therein.

[0009] In another aspect of the invention, the system may further comprise a number of utility modules which supplement or otherwise assist the operations of the security modules. The utility modules may include a user manager module which may further include functionality to edit, create or delete user accounts or templates stored in the system memory. A file removal module may provide for the permanent removal of files from the operational memory. A file may be overwritten with a predetermined pattern such that no trace of the file may be identified. A marking module may provide the functionality to manually mark certain files which are deemed to be critical. This marking function enables the directory check to perform an analysis on this particular file to detect tampering when the directory check module is activated.

[0010] Further functions may be included in the security system to selectively activate particular tools, schedule the automated performance of functions, or provide reports to the system user in a number of different formats.

[0011] Numerous modifications and additions will be apparent to those skilled in the art upon further consideration of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] **FIG. 1** discloses a system diagram for the security system.

[0013] **FIG. 2** discloses a system diagram for the configuration detection subsystem.

[0014] **FIG. 3** discloses a system diagram for the directory checker module.

[0015] **FIG. 4** discloses a system diagram for the user manager module.

[0016] **FIG. 5** discloses a system diagram for the integrity check subsystem.

[0017] **FIG. 6** discloses a system diagram for the network check<module.

[0018] **FIG. 7** discloses a system diagram for the password checking module.

[0019] **FIG. 8** discloses a display graphic presentable on the GUI.

[0020] **FIG. 9** discloses a flow chart which describes the operation of the directory checker module.

[0021] **FIG. 10** discloses a flow chart which describes the operation of the user manager module.

[0022] **FIG. 11** discloses a flow diagram which describes the operation of the integrity check module.

[0023] **FIG. 12** discloses a flow chart which describes the operation of the network check module.

[0024] **FIG. 13** discloses a flow chart which describes the operation of the password checking module.

[0025] **FIG. 14** discloses a flow chart which describes the operation of the file removal module.

[0026] **FIG. 15** discloses a flow chart which describes the operation of the file marking module.

DESCRIPTION OF PREFERRED EMBODIMENTS

[0027] Described herein is an apparatus and method for identifying vulnerabilities and discrepancies in a computer system, and in some situations, suggesting and implementing corrective action. The system disclosed herein is arranged in a modular/integrated form and consists of a number of securities tools and utilities, as well as a number of reporting functions. Each module may test a different aspect of the computer security. The method and apparatus described herein focuses on the internal security of the system, that is, locating security problems that can be detected. The system identifies vulnerable configurations and, in some situations, provides instruction on how to repair particular discrepancies or detected breaches.

[0028] Disclosed in **FIG. 1** is a system diagram which describes a computer system within which the system described herein may operate. In one embodiment of the invention, the computer system may be implemented in a server-type computing device, such as a Unix server with connections to a data network. One connection established to the server may be at least one graphical user interface (GUI) **18** as part of a local area network (LAN). Connections may also be remotely established over the public switched telephone network (PSTN) **9** through a modem device incorporated in the server. The server may also include an Internet connection through which users may establish a connection. The security system described herein may be employed by other remotely located servers which are connected via the data network to the server upon which the security system is resident.

[0029] Returning to **FIG. 1**, the server **10** may include a processor **12** which directs the processes performed by the server. In connection with the processor **12** is an interface device **14** which provides connections to PSTN **9**, data network **16**, and GUI **18**. Although only one GUI is disclosed in the figure, one skilled in the art would know that multiple GUI's may be connected to the server as part of the LAN. The interface may further include a modem device for establishing connections over the PSTN.

[0030] Also, in connection with the processor **12** is the computer system operational memory **13** which contains all the systems directories and files which the security system will perform security operations upon. Also, in connection with the processor are the processing modules **15** which perform the various security, utility, and administrative functions. These modules will be discussed in greater detail below. Finally, during the performance of the various functions certain information may be required in order to perform these processes. This information is stored in database **30**.

[0031] As seen in **Fig. 1** the security system processing modules **15** comprise a number of security and utility modules for performing a variety of operations with regards to the computer system. The following is a brief discussion of the operation of each module.

[0032] In order for the security system to operate on a particular computer system, an analysis of the system must be performed as a preliminary matter. As part of this process, the configuration/setup module **17** identifies files that are "critical" to the computer system and locates any unusual features. This particular module only needs to be operated once upon installation in the computer system. In one aspect of the invention, the configuration/setup module is completely standalone and may not generate a report.

[0033] Disclosed in **FIG. 2** is a system diagram for the configuration/setup module in which the configuration detection subsystem **38**, which is a component of the configuration/setup module, accesses a number of files in the system memory, such as the file system table of contents **40**, the system configuration files **42**, and the system environmental variables **44**. Based on the information accessed, a configuration baseline **46** is generated stored in memory such that it may then be employed by the other modules of the security system.

[0034] During operation of the security system, the directory checker module **18** searches for computer flaws that develop in the file system of a computer over a period of time and detects if "security critical" files have been tampered with. When a particular security problem is identified, the system administrator for the server is prompted for a quick fix, and if the program is capable of providing one, all the information associated with security problems, both corrected and uncorrected, is then forwarded to a reporting module for the security system. Certain things that the

directory check module searches for include: globally read/writable directories, executable files that can be globally modified, protected files that have changed permission, newly created files, protected files that have changed ownership or group, protected files that have been deleted, protected files that have been tampered with, incorrect device driver permissions, tamper device driver permissions, incorrect device ownership, and insecure permissions or ownership of an operating system files.

[**0035**] Disclosed in **FIG. 3** is a system diagram for the directory check module **18**. The directory check module receives data from two sources. The first being the security system database **30** and the second being file system database **42**, which is a listing of files and directories in the system memory including pertinent information relating file or directory ownership, group ownership, and times in which any changes were made to the file or directory. Upon completion of the analysis, a report may be issued via report module **29**.

[**0036**] The directory check module may also examine individual file permissions for nonstandard configurations. System files are compared against the database of suggested permissions for these files. If the files on the computer differ from files in the database, a prompt may be generated to change the files' rights to those suggested by the security system.

[**0037**] The user manager module **20** is employed to identify improper or invalid permissions and ownerships associated with files. The module identifies common misconfigurations and provides reports as to any anomalies detected. The user manager further provides the capability to provide easy access to user account creation, creation of multiple groups, and system wide searches for user account vulnerabilities. The features performed by the module include creation of new accounts, creation of new user groups, searching of home directories for improper ownership, searching for nonexistent home directories, searching home directories for improper groups, and searching home directories for improper or insecure files related to some users.

[**0038**] A system diagram for the user manager **20**, is disclosed in **FIG. 4**. In order to perform its functions, the user manager accesses to a number of different databases. One database is the user list **50**. This list contains a list of all users currently having permission to access the computer system. The default users template **52** contains all of the permissions given to the particular users to access particular files within the system. The file system table of contents **54** includes a listing of all files in the systems with permissions which are granted to each.

[**0039**] The user manager includes the capability to create new user accounts or user groups, as well as make amendments to user templates and file system table of contents. User list **56**, default user template **58** and file system table of contents **60**, are all updated versions of these items following the procedures performed by the user manager. Any items worth noting during the processes performed by the user manager are output via the reporting system **29**.

[**0040**] The integrity checker module **22** performs an analysis of the computer system in order to find security holes located therein. The analysis performed may find vulnerabilities in such things as: the type of computer/operating system used, the access privileges of files, the owner of the files, the group of the files, the date of the files, or a version number for a send mail program. This integrity checker module may provide such items as file name, nature of security hole, and where a system administrator may locate additional information on the particular problems detected. The integrity check module **22** searches for pre-existing security problems by cross-referencing against a vulnerability database which is stored in local memory.

[**0041**] Disclosed in **FIG. 5** is a system diagram which includes the data stores accessed by the integrity check module **22**. As described above, the integrity check module is employed to analyze the computer system and identify vulnerabilities and discrepancies. Data to be analyzed is retrieved from the file system table of contents **72** which includes a listing of files to be analyzed. Also in connection with the integrity module is the vulnerability database **70** which includes a listing of potential vulnerabilities. Items contained in the vulnerability database which are employed when analyzing a filed, may relate to age, owner, permissions, existence and group. Any vulnerabilities or discrepancies detected during the process are output via the report subsystem **29**.

[**0042**] The network check module **24** performs various analysis to detect vulnerabilities which may occur due to a computer or server being connected to a network. The checks which may be performed include: checking vulnerable configuration files, detecting excessive system services, and checking for promiscuous mode operations on the network interface. The network check may display all services running on the network and include those not registered with the Internet Daemon.

[**0043**] Disclosed in **FIG. 6** is a system diagram which includes the elements of the system accessed by the network check module. In order to check the vulnerability of the configuration files, access is gained to the system configuration files **42**. In order to check other system characteristics such as promiscuous mode operations, the operations of the network interface **14** are analyzed. The identification of excessive system services may be determined through analysis of a number of components such as the network interface, the processor, and a number of different files stored in memory. Upon completion of the above-described processes, a report may be issued to the system through employment of reporting system **29**.

[**0044**] The password checking module **26** is employed to examine DES-encrypted passwords associated with each user to locate weak password choices or those easily guessed. This tool may be employed to test the strength of a system front end security, as weak passwords can easily compromise the system. The password checking module may perform such functions as "same salting," integration of "similar salts," filtering of words to generate pseudo words often used as passwords, GCOC password guessing to determine the technique used by the system administrator when handing out new accounts, and large common nonrepetitive dictionaries so that multiple dictionaries that don't contain duplicate words can used for testing.

[**0045**] A system diagram for the password checking module is shown in **FIG. 7**. The module receives data input from two sources. The first is the password file or resume file **80**

which contains all passwords for the users in the system. The second data input is from the word list **82** which includes all of the information to be employed by the password checking module including dictionaries. Output from the module is a list of insecure passwords **83** which are identified from the analysis, as well as a resume file **84**.

[**0046**] The remaining processing modules relate to performing various utility and administrative functions. Under the direction of system user, various files and directories in system memory may be identified and through the use of the modules various functions performed with regards to these items. The user manager **20**, may be employed to generate, delete, or edit user or group directories. Further, the user manager may provide specifics for a selected user, such as user name, UID, group name, GCOC's s-field, home directory, shell, and password. The user manager templates can be used to create user accounts for users who share common requirements on a system.

[**0047**] The file removal module **27** provides functionality to permanently delete selected files. This is accomplished by overwriting the file with bit patterns and text multiple times and then verifying that the information has been changed. This particular function provides the ability to delete individual or groups of files.

[**0048**] The file marking module **28** provides the functionality to manually mark a file which may be critical to the computer system. Through employment of the configuration directory check module described above, certain files may be designated as critical to the system. If there are other files in the system that are critical but not identified as such, then the file marking utility may be employed to mark those files. This causes the file to be checked by the directory check module each time it is run. If the directory check module detects tampering in a marked file, it will be shown in a report for that particular run of the system. This utility may also be employed to unmark files previously marked.

[**0049**] The reporting module **29** provides the functionality to display to a system user, the vulnerabilities and other items generated by the security system. After modules have performed particular functions, reports may be generated which can then be presented to system user via the GUI.

[**0050**] The schedule module **32** provides the functionality to run security checks at predetermined intervals. Checks can be scheduled to run at specific designated times as well as at regular intervals such as monthly or weekly. The schedule module further provides the flexibility to run individual security modules or all tests.

[**0051**] In operation, the security system is initially installed on the computer system. After installation, the configuration setup module **17** will run and perform an evaluation of the computer system. Once this evaluation is complete, this information is stored in memory, and the other modules may be accessed and their functions performed.

[**0052**] In order for the system to perform the functions described herein, a number of different system users interaction devices may be employed. As a first example, a series of screen displays may be presented through the GUI which a system user may interact with in order to activate or deactivate particular functions. Further, options may be provided through the GUI to run individual modules of the system, on all security modules, schedule the operation of the modules, and to receive input from the system user during the operation of the security system. For example, disclosed in **FIG. 7** is an example of a screen display which may be employed to activate the individual modules of the security system.

[**0053**] As can be seen in the display graphic **90**, three separate interface buttons are provided so that a system user may select the modules that will be employed in the analysis of the system. For example, if security button **92** is pressed, the selections enclosed in the dialogue box **98** are presented. As can be seen, these include the directory check, password check, network check, integrity check, and user check functions. User may select the processes to be performed and through selection of the execute button **100** execute these selected functions.

[**0054**] In a situation where the utilities button **96** is selected from the display graphic **90**, the file removal and file marking options will be presented to the system user. Upon selection of the reports section button **94**, the system user may then initiate the performance of reporting or scheduling functions.

[**0055**] If the security button is chosen, the system user may then choose any of the security functions. For example, if the directory check function is chosen in dialogue box **98**, the directory check module **18** is initiated in the system and the steps disclosed in the flowchart of **FIG. 9** are performed.

[**0056**] Once the directory check process has been initiated, the first step is to access files in the file system database. Files selected are typically used files residing in a public binary executable directory or common directories where insecurities may exist. The first step in the process is to access the first file in the file system. At this point, a query is made as to whether the file is a directory or not. If the file is a directory, further queries are made as to whether the file is a new file system, and if so, whether it is traversable. If the answer as is yes to both queries the directory is accessed and the files contained therein may be analyzed. If the new file system is not accessible, the function is terminated. If it is first determined that the directory is an old file system it is accessed and the files contained therein are analyzed.

[**0057**] If the file is not a directory, a query is made as to whether the file is "end of directory". If so, the function is terminated. If the file is not an end of directory, a query is made as to whether the file is a device driver, in the baseline database, or is an executable file. If the answer is "no" to all these queries, this portion of the process is terminated and the next file in sequence is accessed. If a "yes" is determined for any of the queries, an analysis is then performed as to whether the permissions for the file are secure. As was described above, the tests performed include identification of: globally read/writable directories, executable files that can be globally modified, protected files that have changed permission, newly created files, protected files that have changed ownership or group, protected files that have been deleted, protected files that have been tampered with, incorrect device driver permissions, tamper device driver permissions, incorrect device ownership, and insecure permissions or ownership of an operating system files.

[**0058**] If an insecure permission is detected, the system then may provide a report. Depending on the permission problem detected, the system may provide the opportunity to

correct it. These corrections are included as part of the security system database. If permission is given to make the correction to the system, the correction is performed and the process returns to the next file in the file system.

[**0059**] If the system users wishes to initiate the user check function, the steps disclosed in the flow or chart **FIG. 10** are performed. In the initial step, the password file for a the users is first loaded. At this point, the first user on the list is identified. Within the computer system, users may be assigned a home directory in which all files related to or created by the particular user may be stored. A query is first made as to whether the user owns the home directory. If the user does not own the home directory, a report is generated and the process moves on to the next step. A query is then made as to whether the work group to which the user is affiliated owns the home directory. If it is detected that the user's group does not own the home directory to which the user is associated, a report is generated.

[**0060**] In the next step, an analysis is made to determine if the home directory for the user even exists. If this directory does not exist, a report is issued. In the next two steps, an analysis is made as to certain aspects of the user's account and access to the system. In either case, if the permissions provided to the user are found to be insecure, reports are issued. Once the analysis of the particular user is complete, the process returns to the top and the next user on the list is analyzed.

[**0061**] Disclosed in **FIG. 11** is a flow chart which describes the operation of the integrity checker, when selected by the system user or otherwise automatically initiated. The first step in the process is to load the vulnerability database which contains a listing of possible vulnerabilities or discrepancies. The first file from the computer system database is then loaded and the analysis is begun. A first query made is to whether the detected owner of the file matches a predetermined profile. If not, this discrepancy is noted and the next entry in the database is loaded. If the response is yes, in the next step a query is made as to whether the file's group matches a predetermined profile. If not, the next entry in the database is loaded. If the answer is yes, a query is then made as to whether the file permissions match the profile. If they don't, this is noted and the next entry in the database is loaded. But if they do match, a query is made in the final step as to whether the file date predates a match. If the answer is no, the next entry in the database is loaded. If the answer is yes, a report is generated regarding the possible existence of a vulnerability. At the completion of the analysis of the database, a report is generated which lists all discrepancies or vulnerabilities which were noted.

[**0062**] Disclosed in **FIG. 12** is a flow chart which describes the operation of the network checking module when selected by the system user or otherwise automatically initialized. This module is employed to check for vulnerabilities which may occur due to the connection of a server or computer to a data network such as a LAN or the worldwide network. In the first step, an analysis is performed to determine if the system is running in the promiscuous mode. This mode allows the machine to see all network packets transmitted in the network, rather than just those packets destined for the machine. If it is, a report is generated. In the next step, an analysis is performed of the various configuration files to note any insecurities. In the final step, a portscan is performed on all or a designated number of network access ports. Upon completion, a report may be generated and provided.

[**0063**] Disclosed in **FIG. 13** is a flow chart which describes the operation of the password checking module when selected by the system user or otherwise automatically initiated. In the initial step the password information is loaded from the computer system working memory. A query is made as to whether the password file is shadowed, and if so, this file is loaded as well. In the next step, similar salt entries are read from the dictionary stored in the system. The system employs "same salting" so that there will only be a single "salt" attempt per dictionary. After a similar salt entry is chosen, the next **5,000** words from the dictionary are also read. If this is the password's first entry through the system, the GCOS password guessing process is also performed. A word filtering process is then performed to generate pseudowords that are often used as passwords. Once this process is performed, a query is made as to whether the password is in the list of words generated above. If the word is in the list, a query is made as to whether the word from the list is in the password. If the word is in the password, the user is removed from the list. If the word is not in the password, the word is removed from the list.

[**0064**] Once a word is removed from the list, a query is made as to whether there are any words left in the dictionary to employ for the analysis. If yes, the above process is then performed for the words that are left. If the answer is no, it is determined that the password is uncrackable according to the processes described above and a query is made as to whether this is the last user to be analyzed. If the answer is no, then word list is reset to the beginning.

[**0065**] With regards to activating the utility modules, button **94** in the screen display of **FIG. 4** may be selected by a system user. Upon selection of this button, a listing of the utilities modules is provided. The system user may then select one or more utility modules to run.

[**0066**] Disclosed in **FIG. 14** is a flow chart which describes the operation of the file removal module **27**. As described above, this module provides the ability to completely delete selected files so that they are not recoverable. Once a file has been identified for removal by the system user, the file removal process may be initiated. The system user may select a file or files to be deleted by viewing a directory listing on the display screen. Once a file has been selected, an analysis is performed to determine whether this is a file which may be erased. For example if the file passed to the removal module isn't a direct filename (i.e., it contains ".." or "." as a path, possibly to fool the system into wiping out a device instead of a file), the file removal module will not erase the file. If the file is not erasable, the program is terminated.

[**0067**] If it is determined that the file is overwritable, the module then overwrites the file with a specified bit pattern. For example, this pattern may be "**0101**". Once this is complete, the file system is synchronized in order to force data to be written to the drive. The file is reread back to check for differences. At this point a query is made as to whether the file has changed to the designated bit pattern. If not, a report failure to override is provided. If the override was successful, this process may then be repeated a number of times with different bit patterns. This file may then be

overridden with text such as "the quick brown fox jumps over the lazy dog" in order to simulate "non-sensitive" information. The final step in the process is to unlink the file from the volume table of contents. At this point the program may be terminated.

[**0068**] Disclosed in **FIG. 13** is a flow diagram which describes the operation of the file marking module **28** when selected by the system user. As was described above this utility is employed when a file is determined to be critical but is not otherwise marked by one of the security functions. This utility also includes the functionality to unmark files. In the first step the file is selected from memory. As with the file removal module, the system user may view a directory and make selections. The database of the host's security checksum is then loaded. If a file is to be marked, the marked file is added to the end of the checksum file. If it is to removed, it removed from the database. The database is then resaved.

[**0069**] Also, as part of the utility modules, a system user may schedule the performance of any of the functions performed by the security modules or the utility modules. Upon the selection of a schedule option, a variety of further screens may be presented which provide the system user the choices of one or more modules scheduled, the date which the functions will be performed and the time during the dates which they will be performed. Further options may be provided such as periodic activation of the functions, one time activations of the functions, or the combination of various security and utility modules.

[**0070**] Returning again to **FIG. 8**, if the system user selects the reports button **94**, at least one option is provided. A first option may be to generate reports for any individual security module, or a combination of modules. An option may also be provided for archiving and accessing archive reports. In the situation where a system user is to generate a report, certain options may be provided through the graphical interface, as to the format of the reports. The system user may also be provided the opportunity to edit and print reports.

[**0071**] The foregoing description of the present invention has been presented for purposes of illustration and description. Furthermore, the description is not intended to limit the invention to the form disclosed herein. Consequently, variations and modifications commensurate with the above teachings, and the skill or knowledge of the relevant are, within the scope of the present invention. The embodiments described hereinabove are further intended to explain best modes known for practicing the invention and to enable others skilled in the art to utilize the invention in such, or other, embodiments and with various modifications required by the particular applications or uses of the present invention. It is intended that the appended claims be construed to include alternative embodiments to the extent permitted by the prior art.

What is claimed is:

1. A security system for a computer apparatus, wherein said computer apparatus includes a processor and system memory, said security system comprising:

at least one security module which under direction from the processor accesses and analyzes selected portions of the computer apparatus to identify vulnerabilities;

at least one utility module which under the direction from the processor, performs various utility functions with regards to the computer apparatus in response to the identified vulnerabilities; and

a security system memory which contains security information for performing the analysis of the computer apparatus.

2. The security system of claim 1 further including at least one graphical user interface in connection with the computer apparatus through which a system user may direct operations of the security system.

3. The security system of claim 2 further including a reporting module which provides status information to the GUI with regards to operations of the security system.

4. The security system of claim 1 wherein the security modules include at least one of:

a configuration/system module which performs an initial analysis of the computer system acquire configuration information;

a directory checking module which analyzes directories and files in the system memory to determine if security critical files have been tampered with;

a user manager module which analyzes the system memory with regards to improper of invalid permissions given to users of the system for accessing particular files;

an integrity checking module which analyzes files in the system memory to identify system vulnerabilities;

a network checking module which analyzes the computer apparatus to identify vulnerabilities created as a result of the computer apparatus connecting with a data network;

a password checking module which analyzes passwords for users of the computer apparatus to identify vulnerabilities.

5. The security system of claim 4 wherein the utilities modules include at least one of:

said user manager module which includes functionality to perform at least one of:

create a user account, modify the user account, delete the user account, create a user template, edit the user template, and delete the user template;

a file removal module which deletes selected files from the system memory and removes links to the deleted file;

a file marking module which marks selected files; and

a scheduling module which may be employed to schedule any and all of the security modules to perform analysis of the system memory.

6. The security system of claim 2 wherein the computer apparatus comprises a Unix server.

7. The security system of claim 6 wherein the server is connected to a data network.

8. The security system of claim 2 wherein a plurality of interface screens are presented at the GUI for controlling operations of the security system.

9. The security system of claim 4 wherein the system memory comprises a list of known vulnerabilities which may be employed by the integrity checking module.

10. The security system of claim 4 wherein the system memory comprises dictionaries and other tools employed by the password checking module.

11. A method of providing a security assessment for a computer system which includes a system memory, comprising the steps of:

providing a security subsystem in the computer system such that functionality of the security subsystem is directed through a processor for the computer system, wherein the security performs steps comprising:

identifying a configuration of system;

accessing the system memory and performing at least one procedure to provide a security assessment for at least one aspect of the computer system;

as a result of any vulnerabilities discovered in the assessment, identifying corrective measures to be taken with regards to the computer system;

reporting the discovered vulnerability and the identified corrective measures; and

upon receiving an appropriate command, initiating the corrective measures.

12. The method of claim 11 wherein the step of performing at least one procedure to provide a security assessment includes at least one of:

performing an analysis of the directories and files in the system memory to determine if security critical files have been tampered with;

analyzing the system memory with regards to improper or invalid permission given to users of the system for accessing particular files;

analyzing the system memory to identify system vulnerabilities;

analyzing the computer apparatus to identify vulnerabilities created as a result of the computer apparatus connecting to a data network; and

analyzing passwords for users of the computer apparatus to identify vulnerabilities.

13. The method of claim 12 wherein based on the identified vulnerabilities at least one of the following steps are performed:

amending, deleting, or creating user accounts;

amending, deleting, or creating user templates;

deleting selected files from the system memory and removing links to said file;

marking of selected files within the system memory.

14. The method of claim 12 wherein the method of analyzing directories and files comprises the steps of:

accessing individual files in the system memory;

identifying the type of file contained therein;

making a determination as to whether the permissions for the identified file are secure;

if the permissions are not secure, providing a report describing the insecurity;

providing corrections for the detected files which are insecure and initializing corrective action upon receiving direction.

15. The method of claim 12 wherein the step of analyzing the system memory with regards to improper or invalid permissions given to users further comprises the steps of:

performing a check to see if a user owns his or her home directory;

performing a check to see if the user's group owns the home directory;

performing a check to see if user related files are valid; and

performing a check to see if the user's directory exists.

16. The method of claim 12 wherein the step of analyzing files in the system memory to identify system vulnerabilities further comprises the steps of:

providing a vulnerability database which includes a number of identified system vulnerabilities;

accessing the individual files in the system memory;

determining whether the file's owner matches a predetermined profile;

determining whether the file's group matches a predetermined profile;

determining whether the permissions associated with the file match a predetermined profile; and

determining whether the files predate a patch; and

providing a report on any vulnerabilities which may exist in the system memory.

17. The method of claim 12 wherein the step of analyzing the computer apparatus to identify vulnerabilities traded as a result of the computer apparatus connecting with the data network: further comprises the steps of:

checking for insecure configuration files;

checking running of excessive system services; and

checking whether the computer system is running in the promiscuous mode.

18. The method of claim 12 wherein the step of analyzing passwords further comprises the step of:

identifying all passwords for the users of the computer system;

reading the passwords and for each identifying a next similar salt entry;

identifying a next predetermined number of words from the dictionary;

performing a word filtering method with regards to the passwords to add to the word list;

determining whether the word is in the list. If the word is in the list removing the user from the list.

19. The method of claim 11 further comprising the step of displaying result of the security analysis via a graphical user interface.

20. The method of claim 11 wherein the computer system is connected to a data network.

* * * * *

Exhibit A-8

US 20020199122A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2002/0199122 A1**

Davis et al. (43) **Pub. Date: Dec. 26, 2002**

(54) **COMPUTER SECURITY VULNERABILITY ANALYSIS METHODOLOGY**

(76) Inventors: **Lauren B. Davis**, Reisterstown, MD (US); **Hui Men**, Rockville, MD (US)

Correspondence Address:
Benjamin Y. Roca, Office of Patent Counsel
The Johns Hopkins University
Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099 (US)

(21) Appl. No.: **10/177,455**

(22) Filed: **Jun. 21, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/300,178, filed on Jun. 22, 2001. Provisional application No. 60/300,175, filed on Jun. 22, 2001.

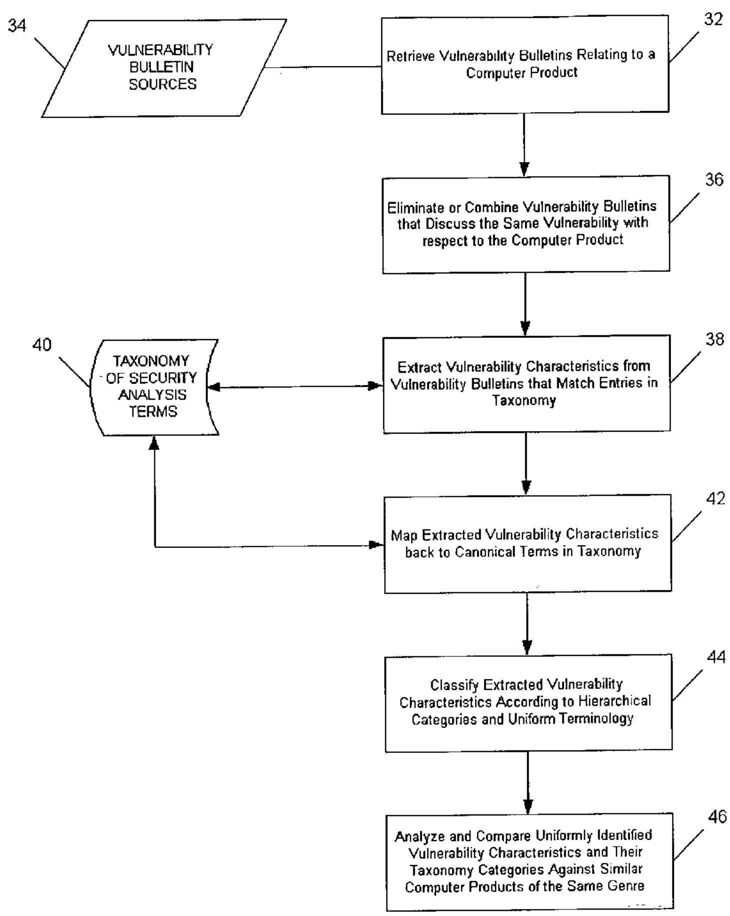
Publication Classification

(51) **Int. Cl.**[7] **G06F 11/30**

(52) **U.S. Cl.** **713/201**

(57) **ABSTRACT**

A methodology of evaluating computer security vulnerabilities in computer products for domain-specific characteristics, statistical trends, and innovative mitigation strategies is presented. The methodology can be programmed into a computer system. Raw security vulnerability data pertaining to a computer product to be analyzed is culled from a pool of trusted resources. Redundant data is combined into separate mutually exclusive records and parsed using a hierarchical taxonomy of security characteristics and security analysis terms. The taxonomy serves to harmonize disparate terminology through the use of canonical terms that equate multiple synonymous terms with the canonical term. The taxonomy also serves to categorize the vulnerability according to a hierarchy of categories and sub-categories so that it may be logically processed and presented to an analyst. Data pertaining to a computer product can be analyzed independently, in composite classes of products, or compared against data that has been similarly obtained and processed for peer products.



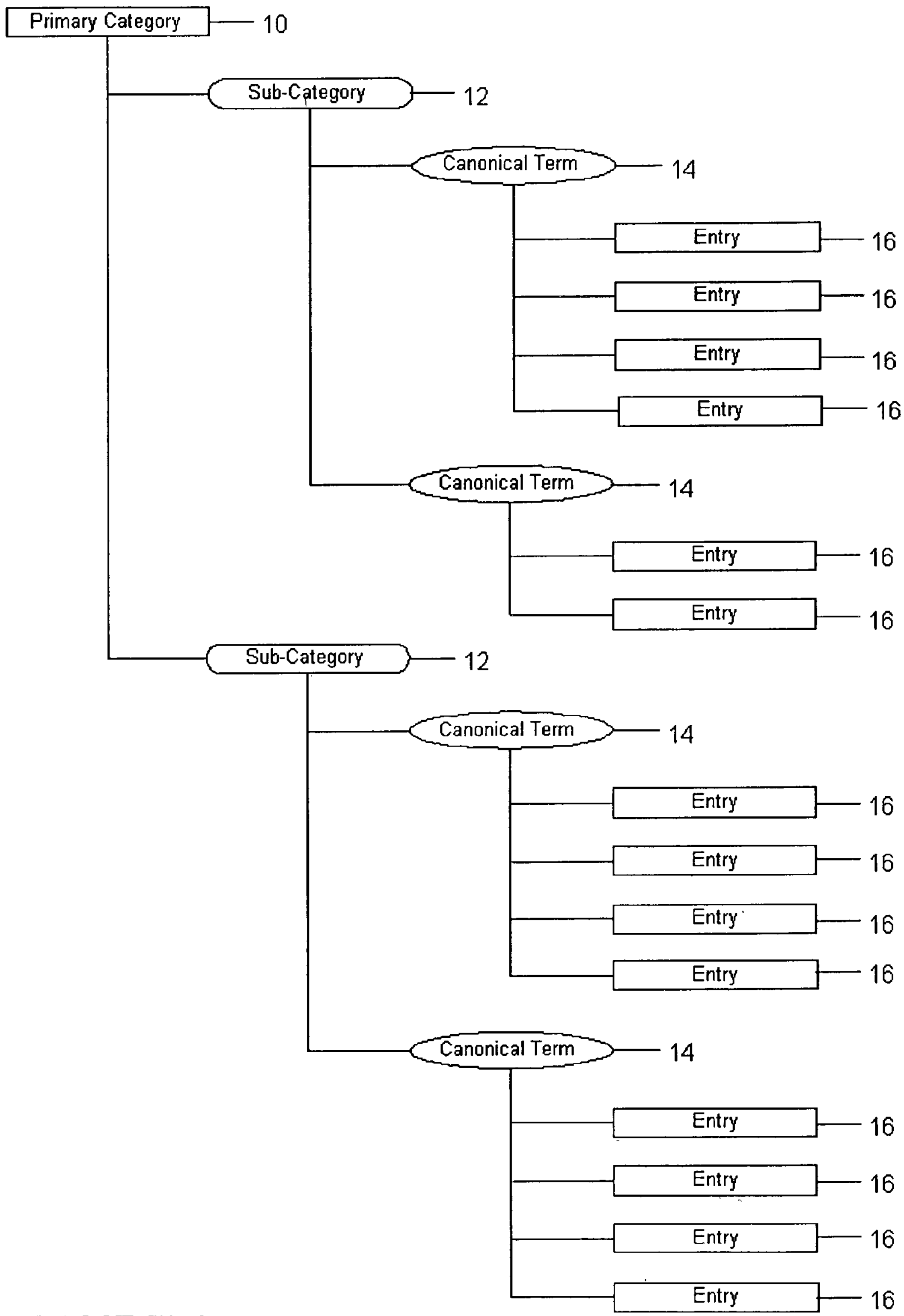

Primary Category
10
Sub-Category
12
Canonical Term
14
Entry
16
Entry
16
Entry
16
Entry
16
Canonical Term
14
Entry
16
Entry
16
Sub-Category
12
Canonical Term
14
Entry
16
Entry
16
Entry
16
Entry
16
Canonical Term
14
Entry
16
Entry
16
Entry
16
Entry
16

FIGURE 1

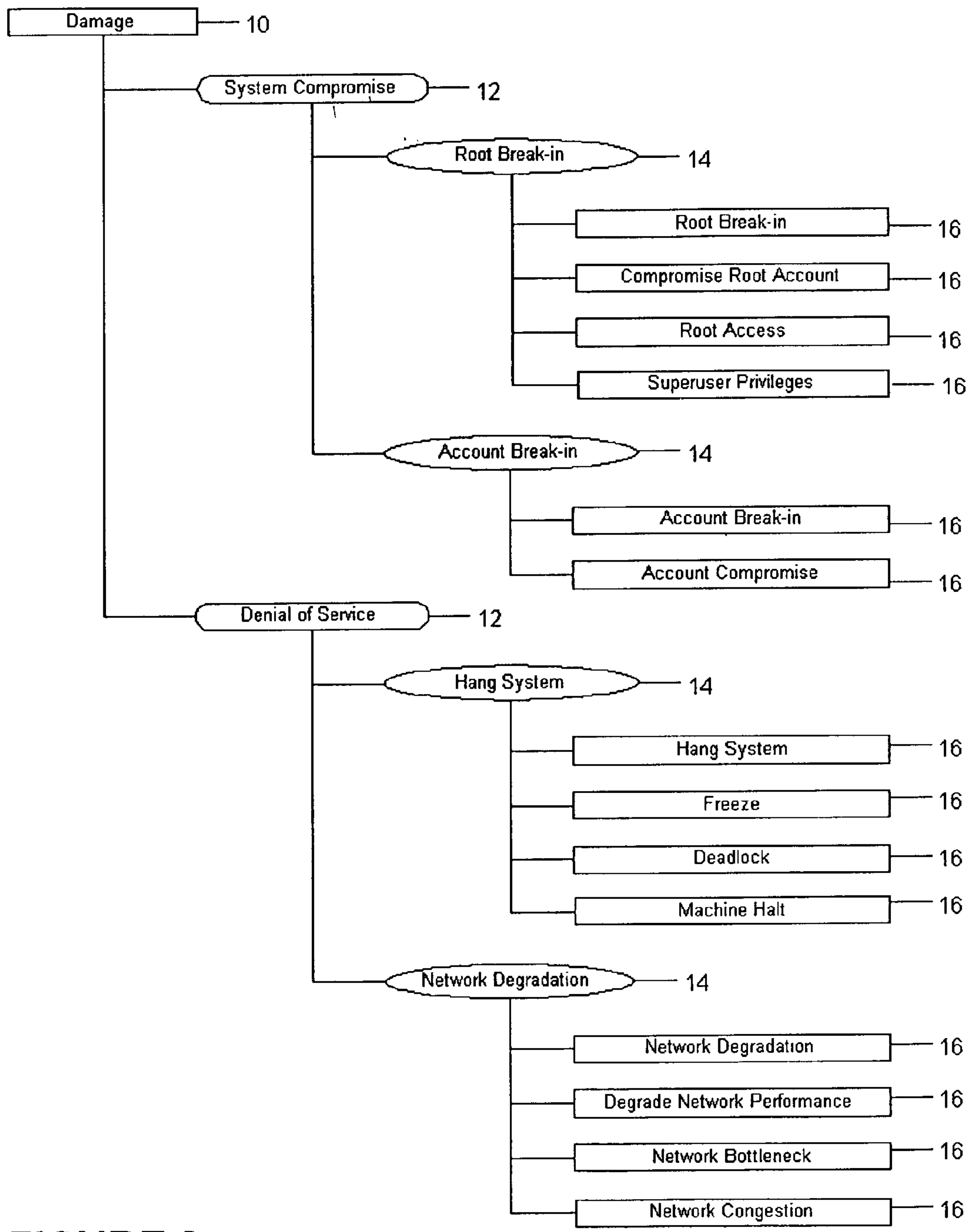

Damage
10
System Compromise
12
Root Break-in
14
Root Break-in
16
Compromise Root Account
16
Root Access
16
Superuser Privileges
16
Account Break-in
14
Account Break-in
16
Account Compromise
16
Denial of Service
12
Hang System
14
Hang System
16
Freeze
16
Deadlock
16
Machine Halt
16
Network Degradation
14
Network Degradation
16
Degrade Network Performance
16
Network Bottleneck
16
Network Congestion
16

FIGURE 2

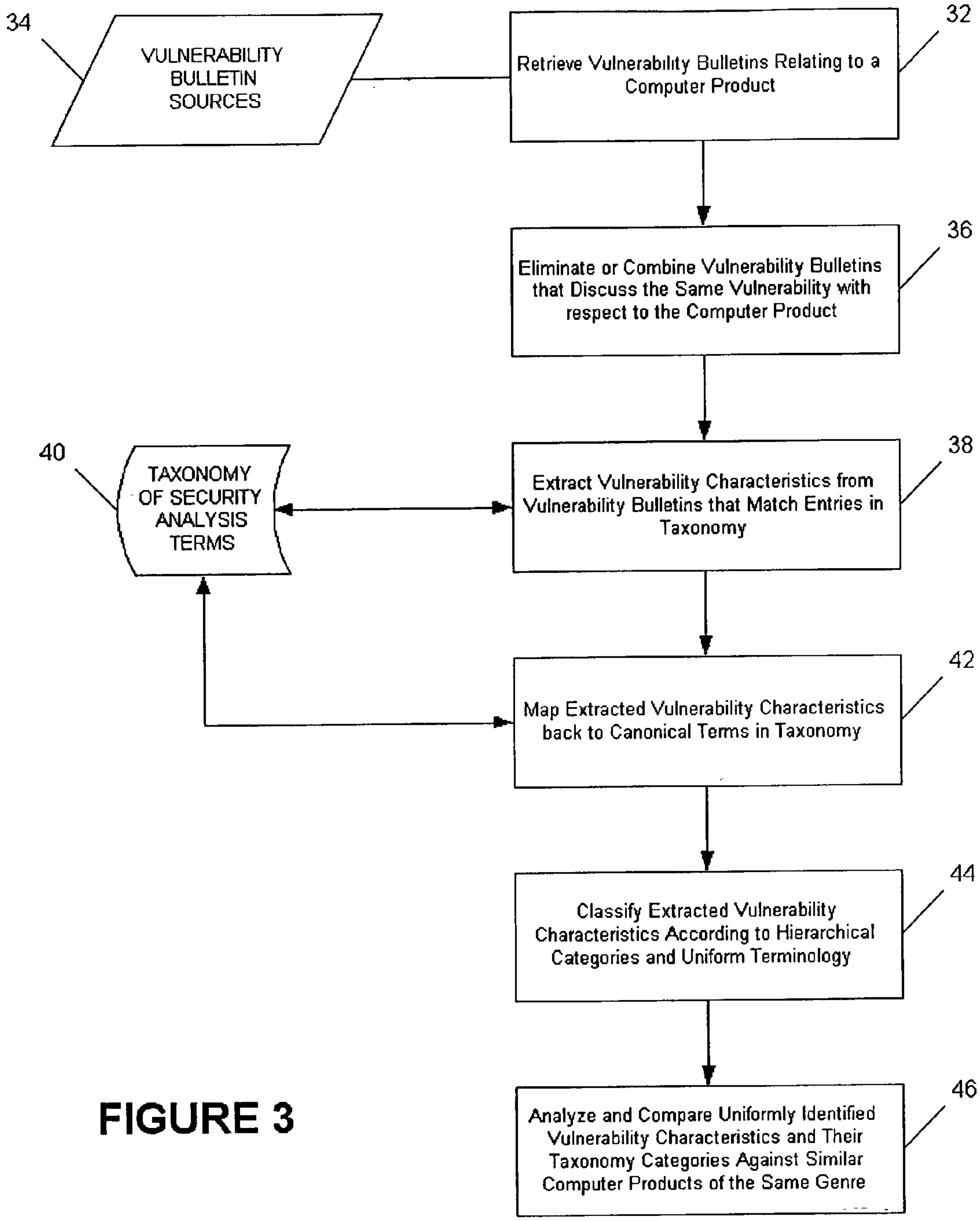
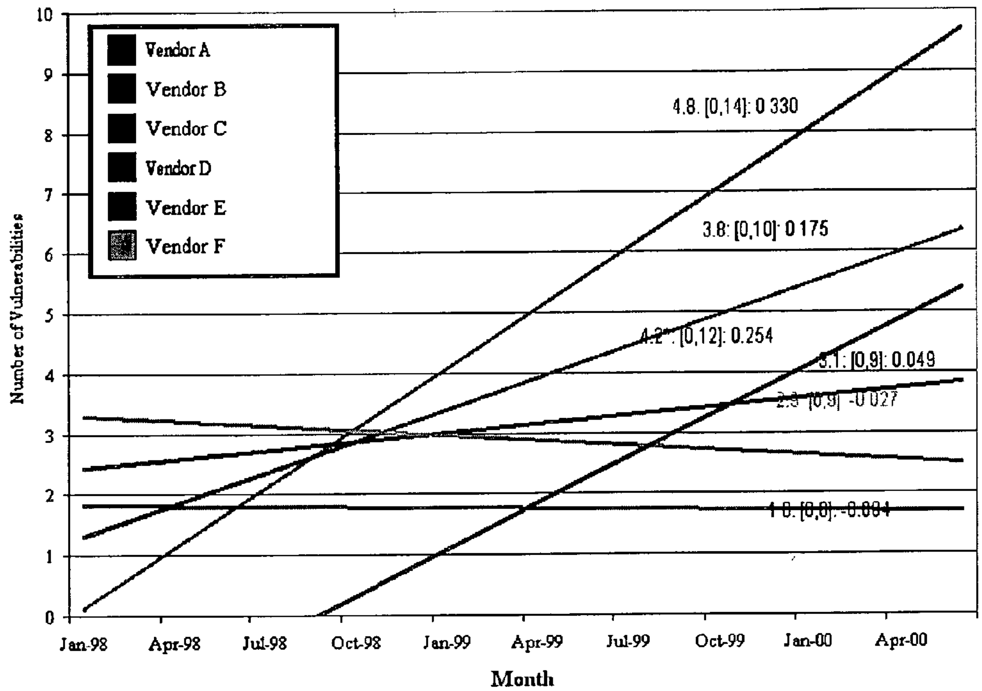

34
VULNERABILITY BULLETIN SOURCES
32
Retrieve Vulnerability Bulletins Relating to a Computer Product
36
Eliminate or Combine Vulnerability Bulletins that Discuss the Same Vulnerability with respect to the Computer Product
40
TAXONOMY OF SECURITY ANALYSIS TERMS
38
Extract Vulnerability Characteristics from Vulnerability Bulletins that Match Entries in Taxonomy
42
Map Extracted Vulnerability Characteristics back to Canonical Terms in Taxonomy
44
Classify Extracted Vulnerability Characteristics According to Hierarchical Categories and Uniform Terminology
46
Analyze and Compare Uniformly Identified Vulnerability Characteristics and Their Taxonomy Categories Against Similar Computer Products of the Same Genre

FIGURE 3

FIGURE 4



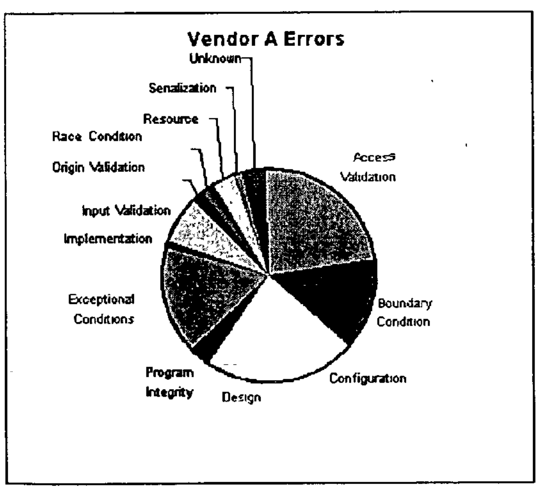


FIGURE 5a

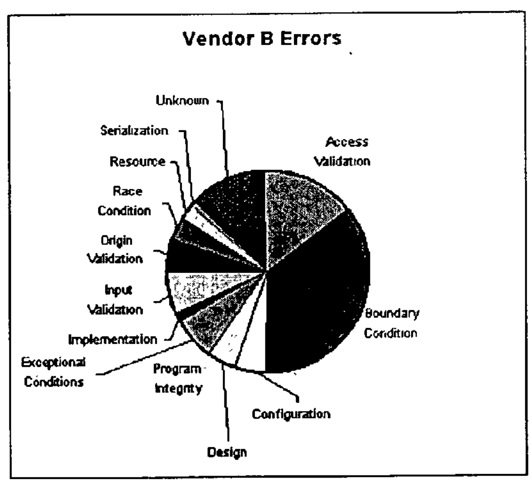


FIGURE 5b

FIGURE 6a

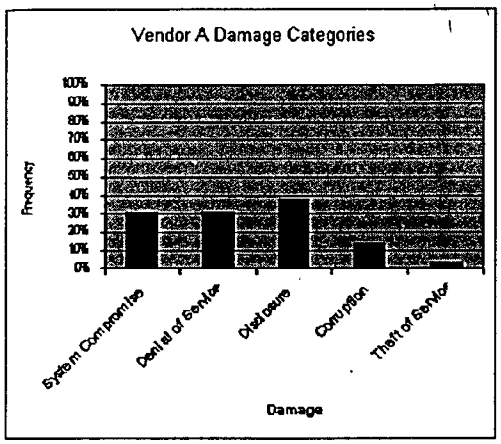


FIGURE 6b

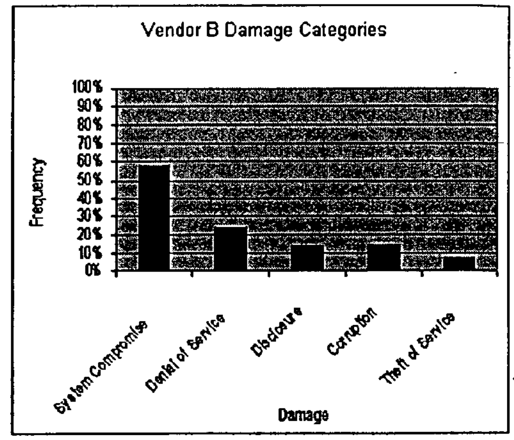


FIGURE 7a

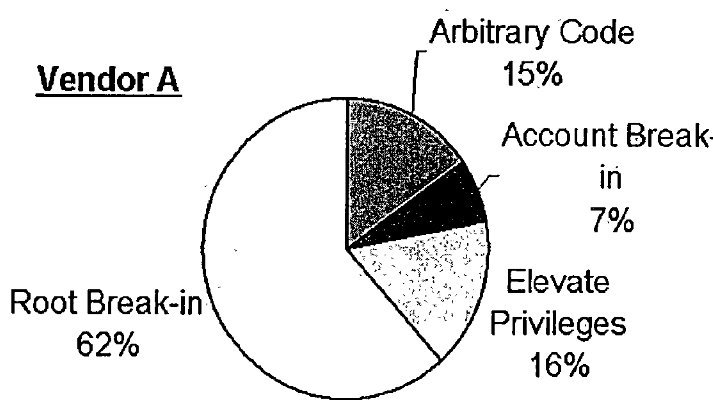


FIGURE 7b

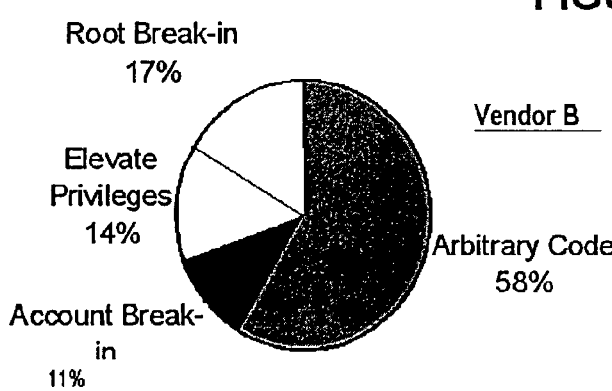


FIGURE 8

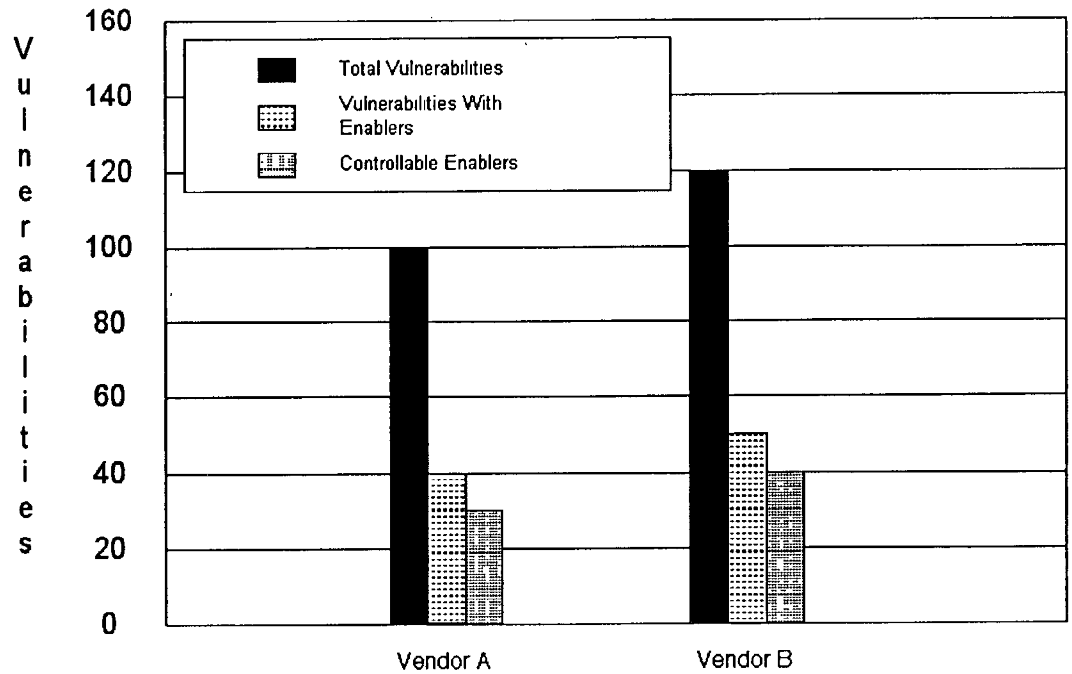
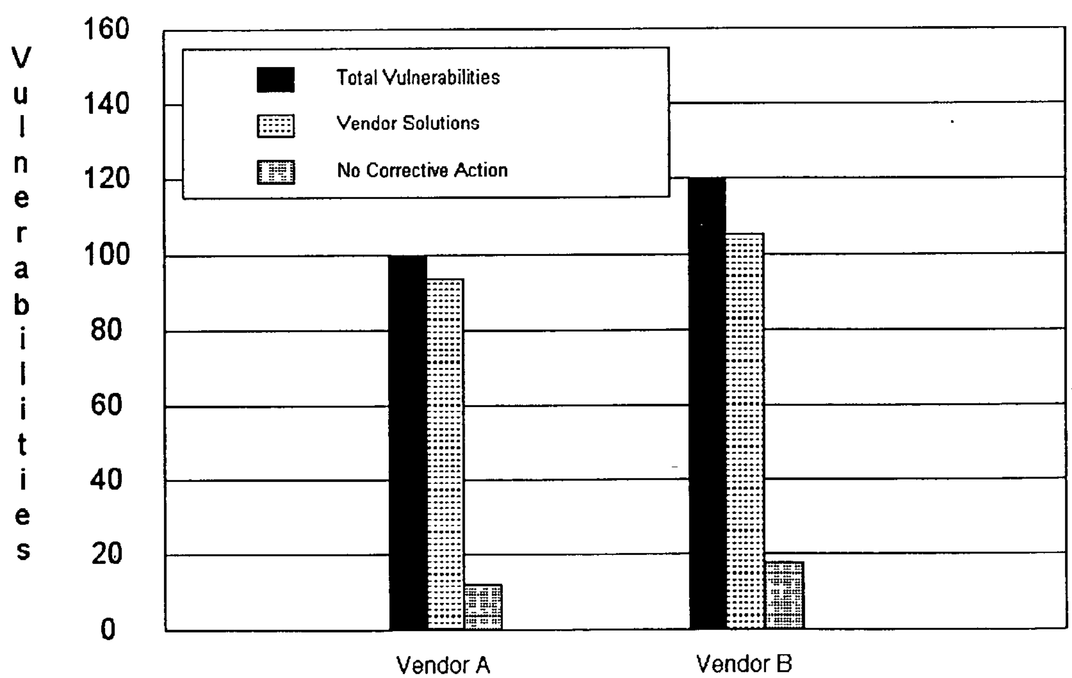


FIGURE 9



COMPUTER SECURITY VULNERABILITY ANALYSIS METHODOLOGY

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 60/300,178, filed on Jun. 22, 2001, which is hereby incorporated by reference in its entirety.

STATEMENT OF GOVERNMENTAL INTEREST

[0002] This invention was made with Government support under contract no. N00024-98-D-8124 with the Department of Defense, Washington, DC. The Government has certain rights in the invention.

BACKGROUND OF THE INVENTION

[0003] Security vulnerabilities in computer products pose a significant concern to computer system users on all levels. The ability to ensure the availability, integrity, and confidentiality of computer systems or at least reduce any damage that may occur as a result of a security vulnerability is of great importance to those responsible for the security of such computer systems.

[0004] Having up-to-date data pertaining to security vulnerabilities of computer products that is presented in an orderly format is essential to creating and operating a computer system resistant to security breaches. Unfortunately, this data is scattered about multiple sources that are not standardized or uniform with respect to terminology, format, or completeness. There currently exists no viable means of organizing reliable security vulnerability data that is scattered about multiple sources into a concise usable format for evaluation of security analysis characteristics and trends.

SUMMARY

[0005] The present invention comprises a methodology for analysis of computer security vulnerabilities for individual computer products, or for classes of computer products such as operating systems, application suites, protocols or information assurance products. The methodology can be programmed into a computer system. Raw security vulnerability data pertaining to a computer product to be analyzed is culled from a pool of trusted resources. Redundant data is combined to create mutually exclusive vulnerability records and applied to a hierarchical taxonomy of security characteristics and security analysis terms. The taxonomy serves to harmonize disparate terminology through the use of canonical terms that equate multiple synonymous terms with the canonical term. The taxonomy also serves to classify or describe the vulnerability according to a hierarchy of categories and sub-categories so that it may be logically processed and presented to an analyst. Data pertaining to a given computer product or class of products may be analyzed as an independent entity or compared against data that has been similarly obtained and processed for peer products in another related class (such as Unix versus Windows operating systems) or specific vendor product comparisons. The comparison provides a basis of evaluation for the given computer product.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] **FIG. 1** illustrates a hierarchical structure of a taxonomy of security analysis terms.

[0007] **FIG. 2** illustrates an example of data in a taxonomy of security analysis terms.

[0008] **FIG. 3** illustrates a flowchart of the analysis of security vulnerabilities for a computer product.

[0009] **FIG. 4** illustrates a vulnerability trend line comparing a computer product against several peer products.

[0010] **FIG. 5***a* illustrates an error analysis for a conglomerate set of operating systems.

[0011] **FIG. 5***b* illustrates an error analysis for a peer conglomerate set of operating systems.

[0012] **FIG. 6***a* illustrates a damage analysis for a conglomerate set of operating systems.

[0013] **FIG. 6***b* illustrates a damage analysis for a peer conglomerate set of operating systems.

[0014] **FIG. 7***a* illustrates a system compromise analysis for a conglomerate set of operating systems.

[0015] **FIG. 7***b* illustrates a system compromise analysis for a peer conglomerate set of operating systems.

[0016] **FIG. 8** illustrates a vulnerability analysis of one type of vulnerability characteristic for a computer product versus a peer product.

[0017] **FIG. 9** illustrates an alternative vulnerability analysis of a different type of vulnerability characteristic for a computer product versus a peer product.

DETAILED DESCRIPTION

[0018] The present invention provides an implementable methodology that can be used to evaluate computer security vulnerabilities of individual computer products, conglomerate sets of computer products, or comparisons of computer products or sets thereof. The term computer product as it relates to the present invention includes computer hardware, computer software, computer firmware, operating systems, protocols, applications, network equipment (e.g., routers, firewalls), and computer peripheral products.

[0019] The present invention relies on two pools of data. The first is a collection of security bulletins from reliable sources with respect to commercial computer products. These sources include, inter alia, *Computer Emergency Response Team* (CERT)-type organizations such as: Carnegie Mellon University's CERT-CC; the Australian Computer Emergency Response Team (AusCERT); the U.S. Department of Energy Computer Incident Advisory Capability (CIAC) Information Bulletins; Internet Security Systems (ISS) X-Force Alerts; Bugtraq Vulnerability Advisories; and specific Vendor Bulletins (e.g., Microsoft, HP, Red Hat, Sun Microsystems, etc . . .). Other security vulnerability data sources may be used at the discretion of an analyst.

[0020] The security vulnerability bulletins are periodically mined for security analysis terms. An example of a vulnerability description that appeared in a June 2000 security bulletin is listed below.

> [0021] ufsrestore Buffer Overflow Vulnerability: Jun. 14, 2000—Boundary Condition Error in ufsrestore affecting Sun Solaris 8.0, Solaris 7.0, and Solaris 2.6, resulting in a local root compromise. The method of

operation of exploitation is via overly long strncat arguments. The setuid properties act as an enabler for exploitation. The recommended corrective actions are to disable the setuid bit, copy utilities to a floppy disk and delete them from the system, and await a forthcoming patch. The risk assigned to this vulnerability is high. Active attacks of this vulnerability were reported at the time the bulletins were issued.

[0022] The second pool of data used in connection with the present invention is a taxonomy of security analysis terms (TSAT), representing security analysis terms that are deemed relevant for the vulnerability analysis, and organized in a hierarchical fashion. Any security analysis terms in the taxonomy that appear in a bulletin are extracted from the bulletin and entered into a spreadsheet or database. The taxonomy is an evolving analysis tool that provides a framework for performing a security vulnerability analysis.

[0023] Combining redundant or overlapping security bulletins creates a mutually exclusive set of vulnerability analysis data. Overlapping security bulletins are not necessarily duplicates, however. They may contain different types of information, but the vulnerability covered may be the same. Consequently, all the information in all the bulletins that pertain to a single vulnerability are included in the resultant spreadsheet or database, but not necessarily as separate entries. Furthermore, multiple bulletins may address a single vulnerability due to independent reporting by numerous organizations and vendors. Or, additional information became available, or further exploits of the vulnerability were detected.

[0024] The taxonomy represents a hierarchical collection of vulnerability characteristic categories and specific vulnerability characteristics within each category, used to describe and classify computer security vulnerabilities. Specific keyword terms are derived from a comprehensive analysis of the reliable sources mentioned above including computer security bulletins, articles, and other security documents. The taxonomy hierarchy is an organization of nested taxonomy categories. The taxonomy is both exhaustive and mutually exclusive.

[0025] The vulnerability characteristics categorized by the taxonomy include: vulnerability error, potential damage resulting from exploitation, severity, enablers, methods of operation, and corrective actions. Taxonomy categories are grouped entities that may contain sub-categories or dictionary entries but not both. Primary categories comprise the base category level in a taxonomy hierarchy. Primary categories may have sub-categories if the primary category is broad enough to be logically partitioned. Similarly, sub-categories may be further decomposed if there exists a logical reason for doing so. Once the lowest level category or sub-category is reached, it is associated with one or more canonical terms.

[0026] A canonical term may be characterized as a standardized description that maps multiple security analysis terms back to a single uniform term. The concept of a canonical term simplifies the analysis process by grouping various different terms or phrases that refer to the same vulnerability characteristic. The use of canonical terms provides a mechanism for reconciling the language employed by different people or organizations when attempting to describe a security vulnerability characteristic. For instance, one bulletin may have labeled potential damage as "Account Break-in" in a description of the computer product vulnerability while another bulletin has labeled the same type of damage as "Account Compromise" in a separate description of the same or similar computer product vulnerability.

[0027] The lowest level in the taxonomy hierarchy is the entry. An entry can comprise words, phrases, non-fixed strings, or full-word strings describing a security analysis term. Every entry is associated with a canonical term. The first entry associated with a canonical term is, by definition, the canonical term.

[0028] **FIG. 1** illustrates a hierarchical structure of a taxonomy. At the root or base level there are primary categories **10**. Sub-categories **12** may exist under the primary categories **10**. Once the hierarchy reaches its lowest categorical level, one or more canonical terms **14** are assigned to the sub-category **12**. The canonical terms are then associated with a list of dictionary entries **16**. Each entry **16** is analogous to the other entries **16** for that category and all of the entries are mapped back to their canonical term **14**.

[0029] It is possible that the primary category **10** need not be partitioned into sub-categories **12** in which case one or more canonical terms **14** are directly associated with a primary category **10**. In addition, a sub-category **12** may be further divided into other sub-categories if there is a logical reason for doing so. Moreover, the number of entries **16** for a canonical term **14** can vary depending on the diversity of the language used to describe a security analysis term. Thus, the hierarchy illustrated in **FIG. 1** is merely an illustration and not intended to limit the present invention.

[0030] **FIG. 2** provides sample data for a taxonomy of security analysis terms. **FIG. 2** has been arbitrarily structured to "read on" the hierarchy presented in **FIG. 1**. The primary category **10** is labeled "Damage". Under the damage category are two sub-categories **12**; System Compromise, and Denial of Service. The System Compromise sub-category **12** is associated with two canonical terms **14** labeled "Root Break-in" and "Account Break-in". The Root Break-in canonical term encompasses four entries **16** in this case. These include Root Break-in, Compromise Root Account, Root Access, and Superuser Privileges. The Account Break-in canonical term encompasses two entries **16** which are Account Break-in and Account Compromise.

[0031] Similarly, the Denial of Service sub-category **12** is associated with two canonical terms **14** labeled "Hang System" and "Network Degradation". The Hang System canonical term encompasses four entries **16** in this case. These include Hang System, Freeze, Deadlock, and Machine Halt. The Network Degradation canonical term also encompasses four entries **16**. These include Network Degradation, Degrade Network Performance, Network Bottleneck, and Network Congestion.

[0032] **FIG. 3** illustrates the methodology used to evaluate computer security vulnerabilities. Security vulnerability bulletins relating to a computer product are retrieved **32** from the pool of trusted sources **34**. Once the relevant security bulletins have been obtained, they are initially reviewed to remove any duplicates **36**. That is, multiple bulletins addressing the same vulnerability characteristic are com-

bined into a single bulletin. Once a mutually exclusive set of vulnerability bulletins pertaining to the computer product has been identified, vulnerability characteristics are extracted from the bulletins **38** by applying the taxonomy **40**. The extracted vulnerability characteristic terms are mapped back to a canonical term in the taxonomy **42**. The mapped terms are then classified according to their hierarchical categories and uniform terminology **44** and entered into a spreadsheet or database. Lastly, a statistical and trend analysis is performed on the terms based upon where the extracted terms fall in the hierarchical categories **46**.

[**0033**] The statistical and trend analysis of the data obtained from the taxonomy comprises the quantification of characteristics of known vulnerabilities. Examples include: a chronology illustrating the frequency of vulnerability reports, the elapsed time between the initial public announcement of a vulnerability and when a vendor solution is issued, the risk of vulnerabilities to exploitation, the types of errors causing the vulnerabilities, the frequency of occurrence as a function of the platform, the scope of damage that can result from exploitation of such vulnerabilities, the actual methods employed to exploit these errors, any corrective actions to remedy the situation, and future projections based on trends documented in available data.

[**0034**] Results of a statistical analysis that can be performed according to the present invention are presented in FIGS. **4**-**9**. These figures illustrate a hypothetical analysis of data for a conglomerate set of operating systems and compares the results against other conglomerate sets of operating systems. The data presented by these examples is fictitious. The purpose of the figures is to illustrate the kind of analysis that can be performed by the methodology of the present invention. The figures comprise charts and diagrams that allow an analyst to evaluate the security vulnerability data for a given computer product, or conglomerate sets of products. The results are presented in terms of a comparison with a peer product or set thereof to help provide a basis for evaluation, but may also be used independently (i.e. noticing that all root break-ins from buffer overflows involve installing a program to always run as root). The example described herein uses only one peer product for comparison purposes. The number of peer products used for an analysis can vary depending on the needs of the analysts and the number of peer products that exist.

[**0035**] **FIG. 4** illustrates vulnerability trend lines for the type of computer product of interest, an operating system. In this example, six operating systems are listed in the analysis. The purpose of this graph is to show a chronology of vulnerability reports for each product. The number strings {w:[x,y]:z} on the graph translate according to the chart:

[**0036**] w: average number of new vulnerabilities reported per month

[**0037**] x: lowest number of new vulnerabilities in any month

[**0038**] y: highest number of new vulnerabilities in any month

[**0039**] z: slope of trend line

[**0040**] Operating systems having steeper slopes indicate more new reported vulnerabilities each subsequent month. This commonly occurs when a product has a rapidly growing user base and or rapidly changing functionality. Products implemented long enough for stability often show a flatter trendline.

[**0041**] Whatever the reason, the illustration in **FIG. 4** provides the analyst with a snapshot of the comparative number of vulnerabilities associated with similar products over time. **FIG. 4** presents vulnerability data analysis in terms of all vulnerabilities, regardless of the type of vulnerability error.

[**0042**] **FIGS. 5***a* and **5***b* present a breakdown of the vulnerability data according to the type of vulnerability error for conglomerate sets of two types of operating systems in the hypothetical example. The data is presented in the form of a pie chart in this example. A cursory examination reveals that Vendor A is susceptible to many more "exceptional condition" errors than Vendor B but produces significantly less "boundary condition" errors than Vendor B. This type of data may be important to an analyst evaluating computer products in regard to the mitigation strategies that might apply to specific types of vulnerability errors.

[**0043**] **FIGS. 6***a* and **6***b* provide a detailed analysis based upon the damage categories of the taxonomy. **FIG. 6***a* plots the percent of vulnerabilities resulting in a particular type of damage category for Vendor A's product. **FIG. 6***b* presents the exact same data for Vendor B's product. The two graphs could have been merged into a single chart if desired. System compromise is the most egregious type of damage. It becomes clear that the percent of vulnerabilities that are severely damaging is greater for Vendor B (approximately 60%) than for Vendor A (approximately 30%).

[**0044**] **FIGS. 7***a* and **7***b* break down the analysis even further by focusing on the subcategories of system compromise specifically. These pie charts list the canonical terms associated with the sub-category of system compromise. **FIG. 7***a* (Vendor A) has a significantly higher occurrence of root break-ins than **FIG. 7***b* (Vendor B). Again, this could be critical information because root break-ins are deemed very serious because of the potential widespread damage that can occur as a result.

[**0045**] **FIG. 8** charts a comparison of Vendor A vs. Vendor B with respect to total vulnerabilities, enablers, and controllable enablers. An enabler is a condition that can affect a particular vulnerability. Some vulnerabilities may require the presence of an enabler to fully exploit the vulnerability. In such cases the vulnerability may be controllable by controlling the enabler as a form of corrective action. **FIG. 8** decomposes total vulnerabilities into vulnerabilities that require enablers and within that subset, enablers that can be controlled. The specific data illustrated in **FIG. 8** reveals that approximately ⅓of the total vulnerabilities for Vendor A and Vendor B require enablers. Moreover, about 80% of the vulnerabilities that have enablers have controllable enablers for the operating system of both vendors.

[**0046**] **FIG. 9** illustrates the number of different types of vendor solutions attributable to the total number of vulnerabilities and the number of vulnerabilities having no corrective action as yet. This data provides an analyst with a sense of whether the vulnerability can be worked around or if it still poses a threat.

[**0047**] The above charts, graphs, and figures for the fictitious example represent data culled from reliable sources

and applied to the hierarchical taxonomy. The breadth and scope of the statistical analysis provides analysts with a wealth of information to be used in considering the types of mitigation strategies to employ for specific products or classes of products, and may be used in evaluation of specific products for system integration.

[0048] To evaluate a computer product against peer products it is necessary to have analyzed the peer products in the same manner as the computer product in question. It is also recommended that the steps that involve retrieving and processing vulnerability characteristics from security bulletins be updated frequently. This ensures that a product is being evaluated with the most recent data available.

[0049] The data from a previous analysis can be archived for future use so that future analysis efforts need not be completely duplicated, merely updated. Archived computer product analyses may need to be updated if they are deemed out-of-date. Updating an analysis entails retrieving security vulnerability data from the present back to the last known date that data was gathered for the computer product in question.

[0050] In addition, from time to time it may be necessary to update the taxonomy to accommodate new categories or newly discovered vulnerability characteristics. New entries may need to be incorporated into the taxonomy and associated with a canonical term. New canonical terms may also need to be created if a new category or sub-category is introduced. Thus, the taxonomy is an evolving tool.

[0051] It is to be understood that the present invention illustrated herein is readily implementable by those of ordinary skill in the art as a computer program product having a medium with computer program(s) embodied thereon. The computer program product is capable of being loaded and executed on the appropriate computer processing device(s) in order to carry out the method or process steps described. Appropriate computer program code in combination with hardware implements many of the elements of the present invention. This computer code is typically stored on removable storage media. This removable storage media includes, but is not limited to, a diskette, standard CD, pocket CD, zip disk, or mini zip disk. Additionally, the computer program code can be transferred to the appropriate hardware over some type of data network.

[0052] The present invention has been described, in part, with reference to flowcharts or logic flow diagrams. It will be understood that each block of the flowchart diagrams or logic flow diagrams, and combinations of blocks in the flowchart diagrams or logic flow diagrams, can be implemented by computer program instructions. These computer program instructions may be loaded onto a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions which execute on the computer or other programmable data processing apparatus create means for implementing the functions specified in the flowchart block or blocks or logic flow diagrams.

[0053] These computer program instructions may also be stored in a computer-readable memory that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory produce an article of manufacture including instruction means which implement the function specified in the flowchart blocks or logic flow diagrams. The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide steps for implementing the functions specified in the flowchart blocks or logic flow diagrams. Accordingly, block(s) of flowchart diagrams and/or logic flow diagrams support combinations of means for performing the specified functions, combinations of steps for performing the specified functions and program instruction means for performing the specified functions. It will also be understood that each block of flowchart diagrams and/or logic flow diagrams, and combinations of blocks in flowchart diagrams and/or logic flow diagrams can be implemented by special purpose hardware-based computer systems that perform the specified functions or steps, or combinations of special purpose hardware and computer instructions.

[0054] In the following claims, any means-plus-function clauses are intended to cover the structures described herein as performing the recited function and not only structural equivalents but also equivalent structures. Therefore, it is to be understood that the foregoing is illustrative of the present invention and is not to be construed as limited to the specific embodiments disclosed, and that modifications to the disclosed embodiments, as well as other embodiments, are intended to be included within the scope of the appended claims. The invention is defined by the following claims, with equivalents of the claims to be included therein.

1. A computer for analyzing security vulnerabilities in a computer product, comprising:

a memory containing:

a retrieval computer program that retrieves computer security vulnerability data pertaining to the computer product being analyzed;

a extraction computer program that extracts vulnerability terms from the retrieved computer security vulnerability data;

a classification computer program that classifies the extracted vulnerability terms according to a hierarchical taxonomy of vulnerability characteristics; and

an analysis computer program that analyzes the classified vulnerability terms and characteristics for the computer product being analyzed, the analysis being based on the taxonomy hierarchy associated with the vulnerability terms; and

a processor for executing the retrieval computer program, extraction computer program, classification computer program, and analysis computer program.

2. The computer of claim 1 wherein the extraction computer program eliminates any redundant data retrieved by the retrieval computer program to create mutually exclusive vulnerability data pertaining to the computer product being analyzed.

3. The computer of claim 2 wherein the classification program associates each extracted vulnerability term for the computer product being analyzed to a canonical term that is

linked with a vulnerability characteristic appearing in the hierarchical taxonomy of vulnerability characteristics.

4. The computer of claim 3 wherein the analysis computer program:

performs a statistical analysis on the classified vulnerability characteristics for the computer product being analyzed; and

organizes the statistical analysis of the vulnerability characteristics for the computer product being analyzed.

5. The computer of claim 4 wherein the analysis computer program further outputs the organized statistical analysis in a human readable format.

6. A method of analyzing security vulnerabilities in a computer product, comprising:

retrieving computer security vulnerability data pertaining to the computer product being analyzed;

extracting vulnerability terms from the retrieved computer security vulnerability data;

classifying the extracted vulnerability terms according to a hierarchical taxonomy of vulnerability characteristics; and

analyzing the classified vulnerability terms and characteristics for the computer product being analyzed, the analysis being based on the taxonomy categories associated with the vulnerability terms.

7. The method of claim 6 wherein the extracting step further comprises eliminating any redundant data retrieved during the retrieving step to create mutually exclusive vulnerability data pertaining to the computer product being analyzed.

8. The method of claim 7 wherein the classifying step further comprises associating each extracted vulnerability term for the computer product being analyzed to a canonical term that is linked with a vulnerability characteristic appearing in the hierarchical taxonomy of vulnerability characteristics.

9. The method of claim 8 wherein the analyzing step further comprises:

performing a statistical analysis on the classified vulnerability characteristics for the computer product being analyzed; and

organizing the statistical analysis of the vulnerability characteristics for the computer product being analyzed.

10. The method of claim 9 wherein the analyzing step further comprises outputting the organized statistical analysis in a human readable format.

11. A computer-readable medium whose contents cause a computer system to analyze security vulnerabilities in a computer product, the computer system having a retrieval computer program, an extraction computer program, a classification computer program, and an analysis computer program with functions for invocation, by performing the steps of:

retrieving computer security vulnerability data pertaining to the computer product being analyzed;

extracting vulnerability terms from the retrieved computer security vulnerability data;

classifying the extracted vulnerability terms according to a hierarchical taxonomy of vulnerability characteristics; and

analyzing the classified vulnerability terms and characteristics for the computer product being analyzed, the analysis being based on the taxonomy categories associated with the vulnerability terms.

12. The computer-readable medium of claim 11 wherein the extracting step further comprises eliminating any redundant data retrieved during the retrieving step to create mutually exclusive vulnerability data pertaining to the computer product being analyzed.

13. The computer-readable medium of claim 12 wherein the classifying step further comprises associating each extracted vulnerability term for the computer product being analyzed to a canonical term that is linked with a vulnerability characteristic appearing in the hierarchical taxonomy of vulnerability characteristics.

14. The computer-readable medium of claim 13 wherein the analyzing step further comprises:

performing a statistical analysis on the classified vulnerability characteristics for the computer product being analyzed; and

organizing the statistical analysis of the vulnerability characteristics for the computer product being analyzed.

15. The computer-readable medium of claim 14 wherein the analyzing step further comprises outputting the organized statistical analysis in a human readable format.

16. A computer system for analyzing security vulnerabilities in a computer product, comprising:

means for retrieving computer security vulnerability data pertaining to the computer product being analyzed;

means for extracting vulnerability terms from the retrieved computer security vulnerability data;

means for classifying the extracted vulnerability terms according to a hierarchical taxonomy of vulnerability characteristics; and

means for analyzing the classified vulnerability terms and characteristics for the computer product being analyzed, the analysis being based on the taxonomy categories associated with the vulnerability terms.

17. The computer system of claim 16 wherein the means for extracting further comprises means for eliminating any redundant data retrieved by the means for retrieving to create mutually exclusive vulnerability data pertaining to the computer product being analyzed.

18. The computer system of claim 17 wherein the means for classifying further comprises means for associating each extracted vulnerability term for the computer product being analyzed to a canonical term that is linked with a vulnerability characteristic appearing in the hierarchical taxonomy of vulnerability characteristics.

19. The computer system of claim 18 wherein the means for analyzing further comprises:

means for performing a statistical analysis on the classified vulnerability characteristics for the computer product being analyzed; and

means for organizing the statistical analysis of the vulnerability characteristics for the computer product being analyzed.

20. The computer system of claim 19 wherein the means for analyzing further comprises means for outputting the organized statistical analysis in a human readable format.

* * * * *

Exhibit A-9

US 20030014669A1

(19) United States
(12) Patent Application Publication (10) Pub. No.: US 2003/0014669 A1
Caceres et al. (43) Pub. Date: Jan. 16, 2003

(54) AUTOMATED COMPUTER SYSTEM SECURITY COMPROMISE

(76) Inventors: **Maximiliano Gerardo Caceres**, Buenos Aires (AR); **Gerardo Gabriel Richarte**, Buenos Aires (AR); **Agustin Azubel Friedman**, Buenos Aires (AR); **Ricardo Quesada**, Buenos Aires (AR); **Luciano Notarfrancesco**, Buenos Aires (AR); **Oliver Friederichs**, Menlo Park, CA (US); **Javier Burroni**, Buenos Aires (AR); **Gustavo Ajzenman**, Buenos Aires (AR); **Gabriel Becedillas**, Buenos Aires (AR); **Bruce Leidl**, Buenos Aires (AR)

Correspondence Address:
FITZPATRICK CELLA HARPER & SCINTO
30 ROCKEFELLER PLAZA
NEW YORK, NY 10112 (US)

(21) Appl. No.: **10/054,307**

(22) Filed: **Jan. 22, 2002**

Related U.S. Application Data

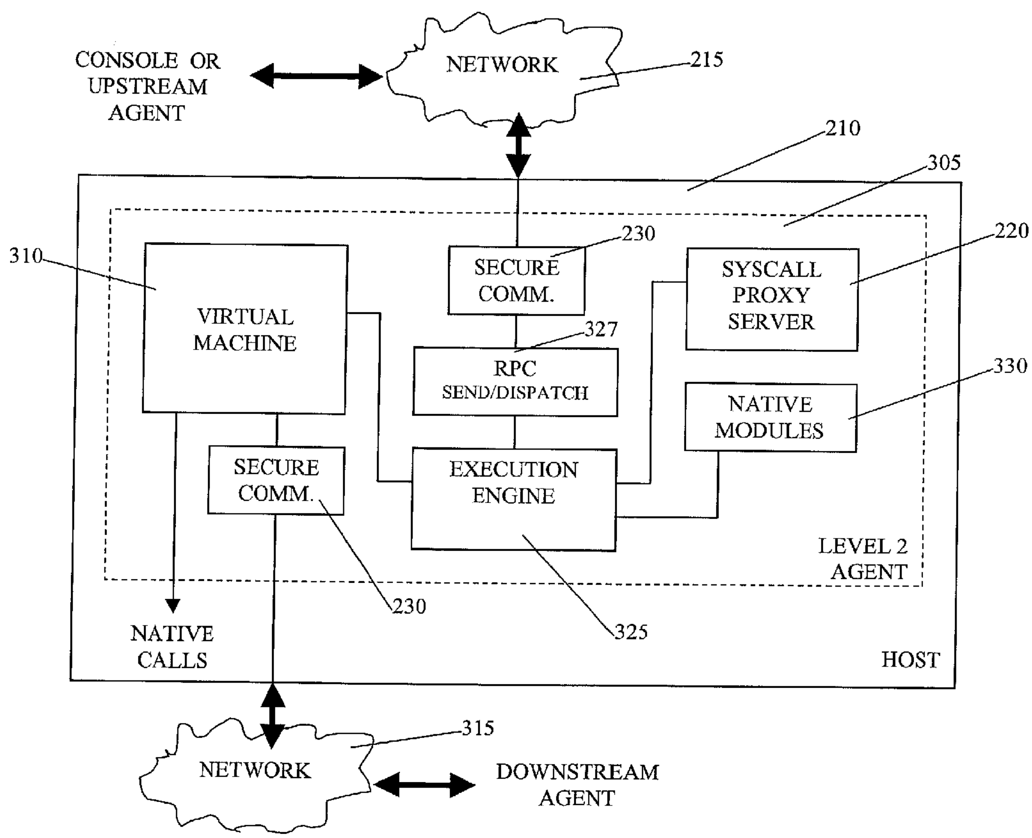
(60) Provisional application No. 60/304,270, filed on Jul. 10, 2001. Provisional application No. 60/313,793, filed on Aug. 20, 2001.

Publication Classification

(51) **Int. Cl.**[7] **H04L 9/00**
(52) **U.S. Cl.** **713/201**

(57) **ABSTRACT**

A system is provided for performing penetration testing of a target computer network by installing a remote agent in the target computer network. The system includes a local agent provided in a computer console and configured to receive and execute commands. A user interface is provided in the console and configured to send commands to and receive information from the local agent, process the information, and present the processed information. A database is configured to store the information received from the local agent. A network interface is connected to the local agent and configured to communicate with the remote agent installed in the target computer network via a network. Security vulnerability exploitation modules are provided for execution by the local agent and/or the remote agent.



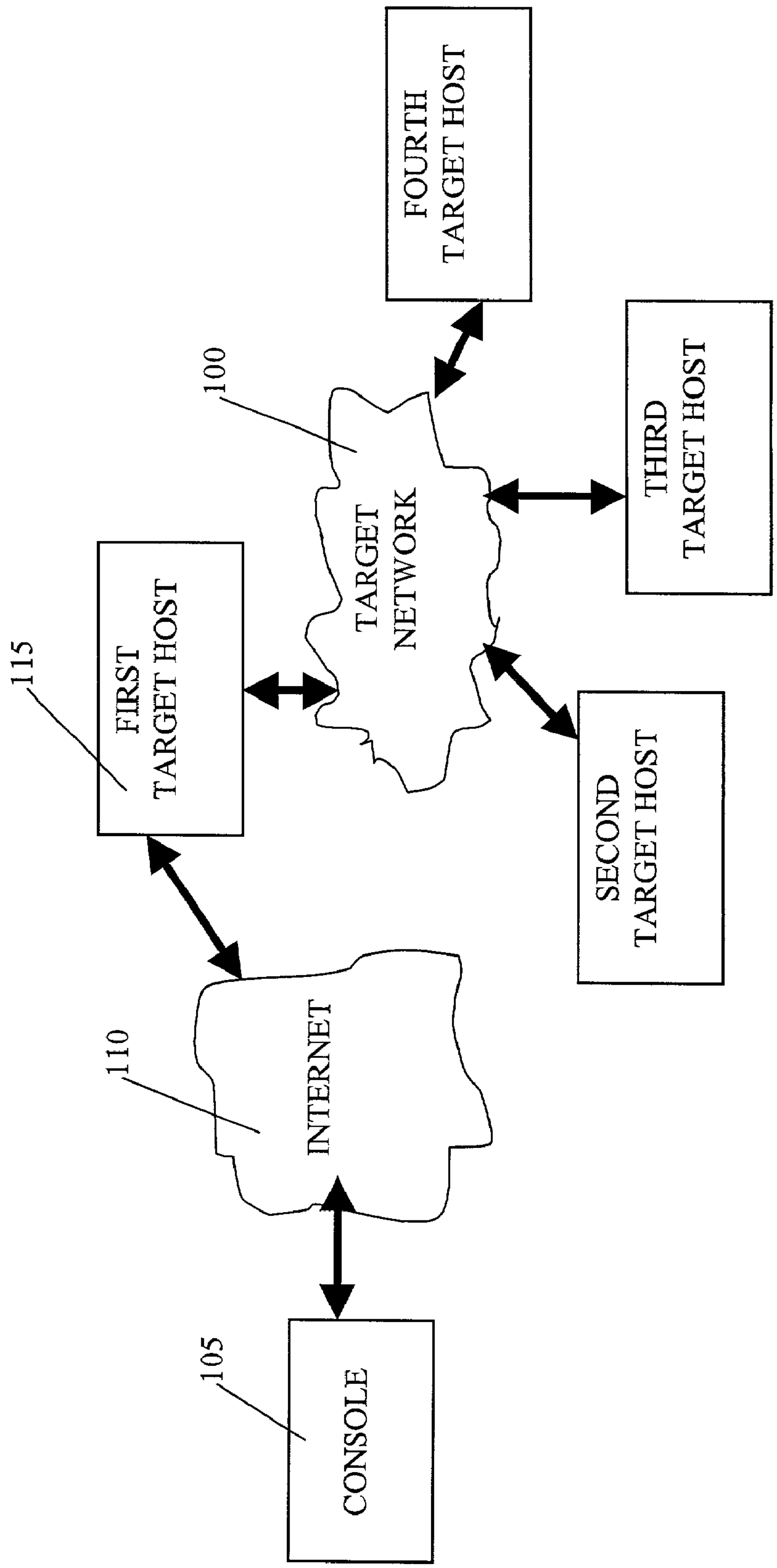

FOURTH TARGET HOST
THIRD TARGET HOST
100
TARGET NETWORK
FIRST TARGET HOST
115
SECOND TARGET HOST
110
INTERNET
105
CONSOLE

Fig. 1

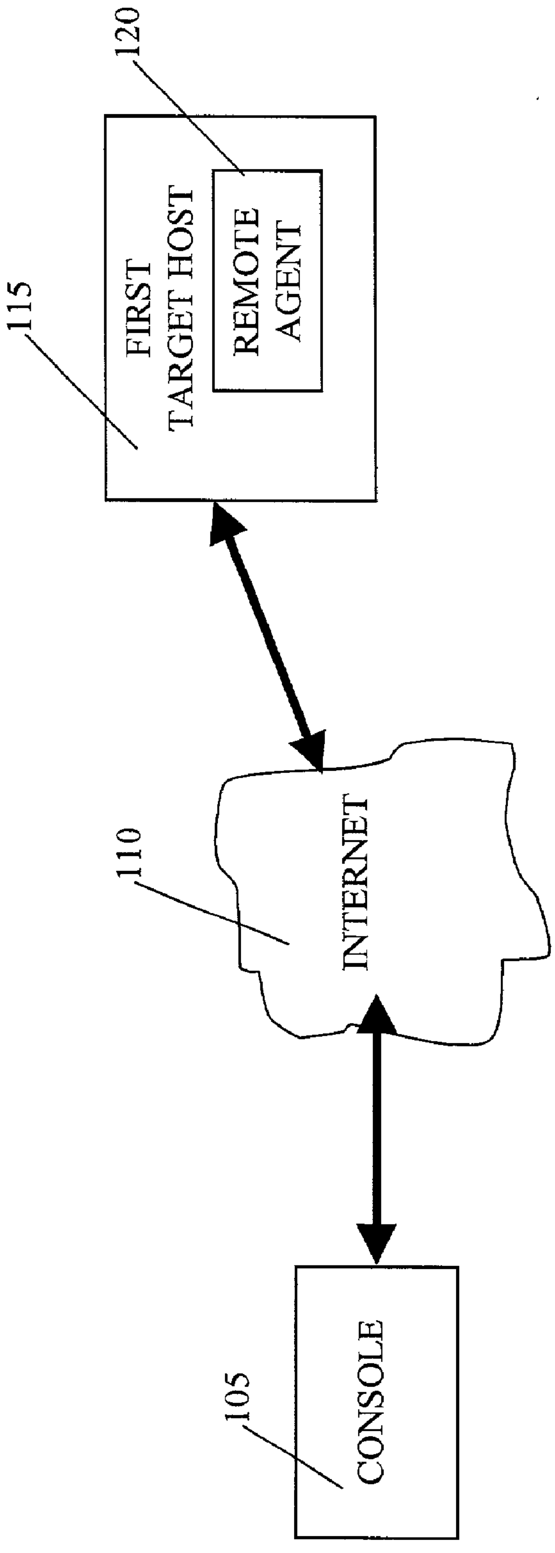

120
115
FIRST
TARGET HOST
REMOTE
AGENT
110
INTERNET
105
CONSOLE

Fig. 2

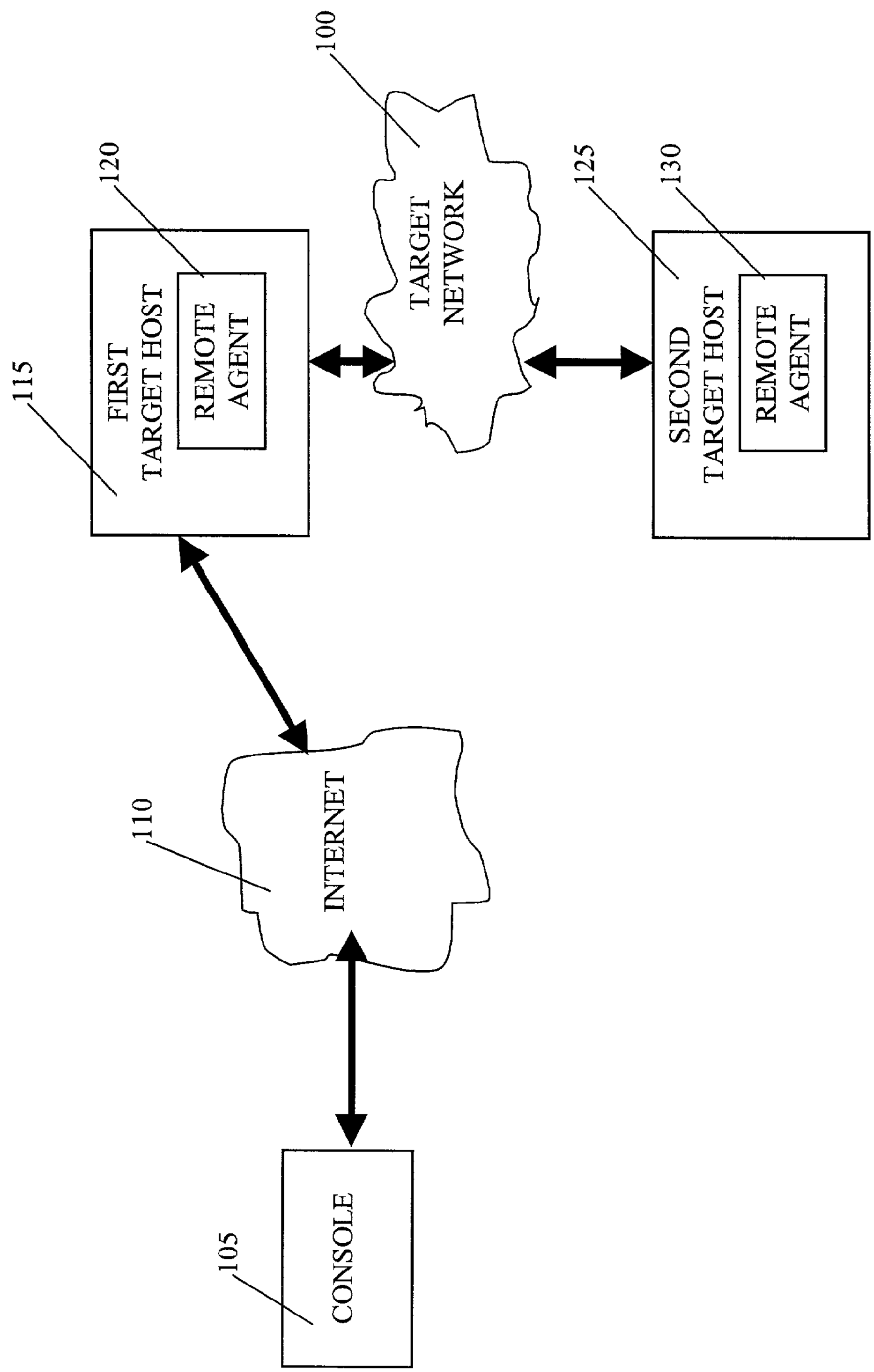

100
TARGET NETWORK
115
FIRST TARGET HOST
120
REMOTE AGENT
125
SECOND TARGET HOST
130
REMOTE AGENT
110
INTERNET
105
CONSOLE

Fig. 3

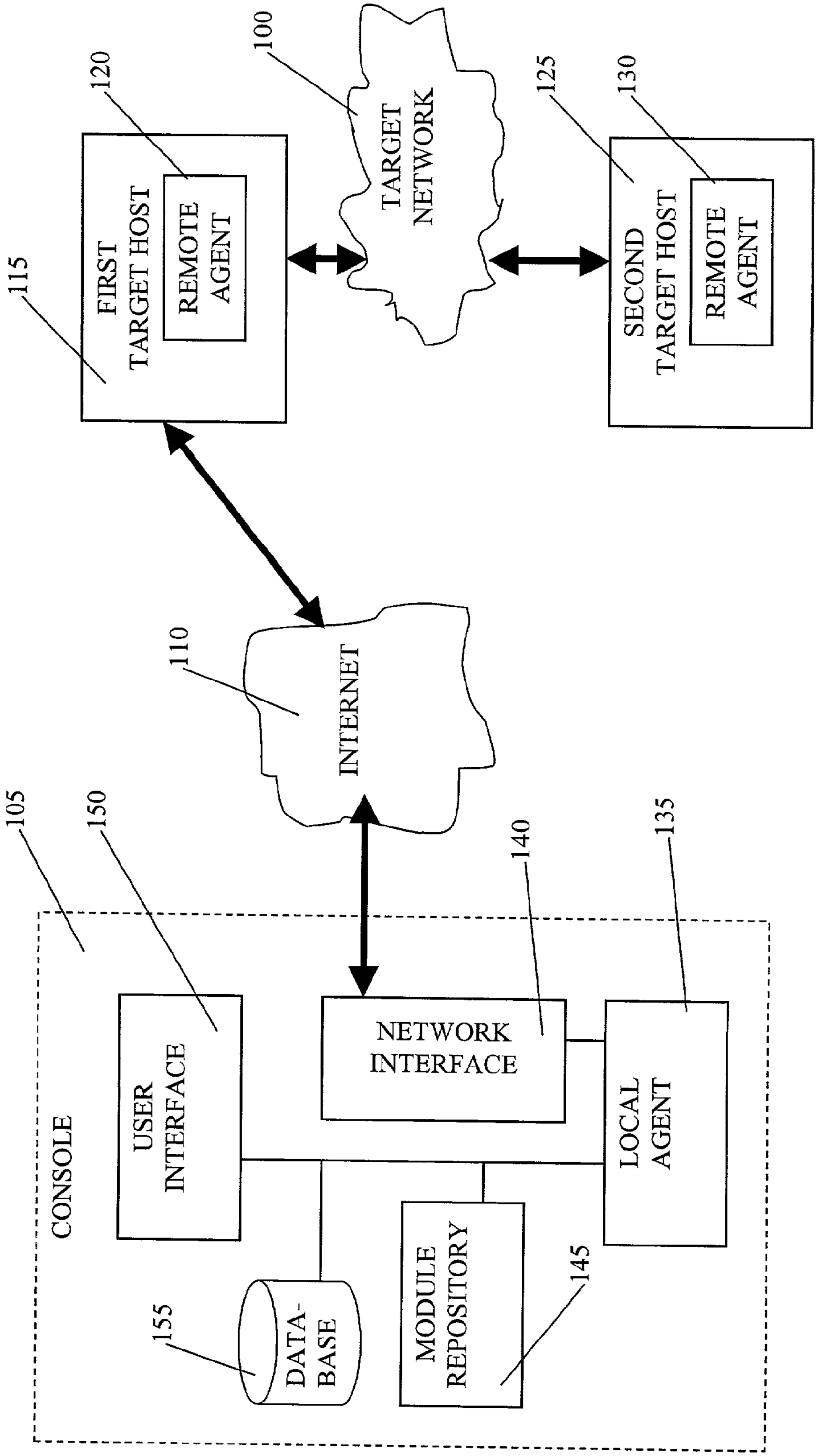

FIRST TARGET HOST
REMOTE AGENT
TARGET NETWORK
SECOND TARGET HOST
REMOTE AGENT
INTERNET
CONSOLE
USER INTERFACE
NETWORK INTERFACE
LOCAL AGENT
DATA-BASE
MODULE REPOSITORY
100
105
110
115
120
125
130
135
140
145
150
155

Fig. 4

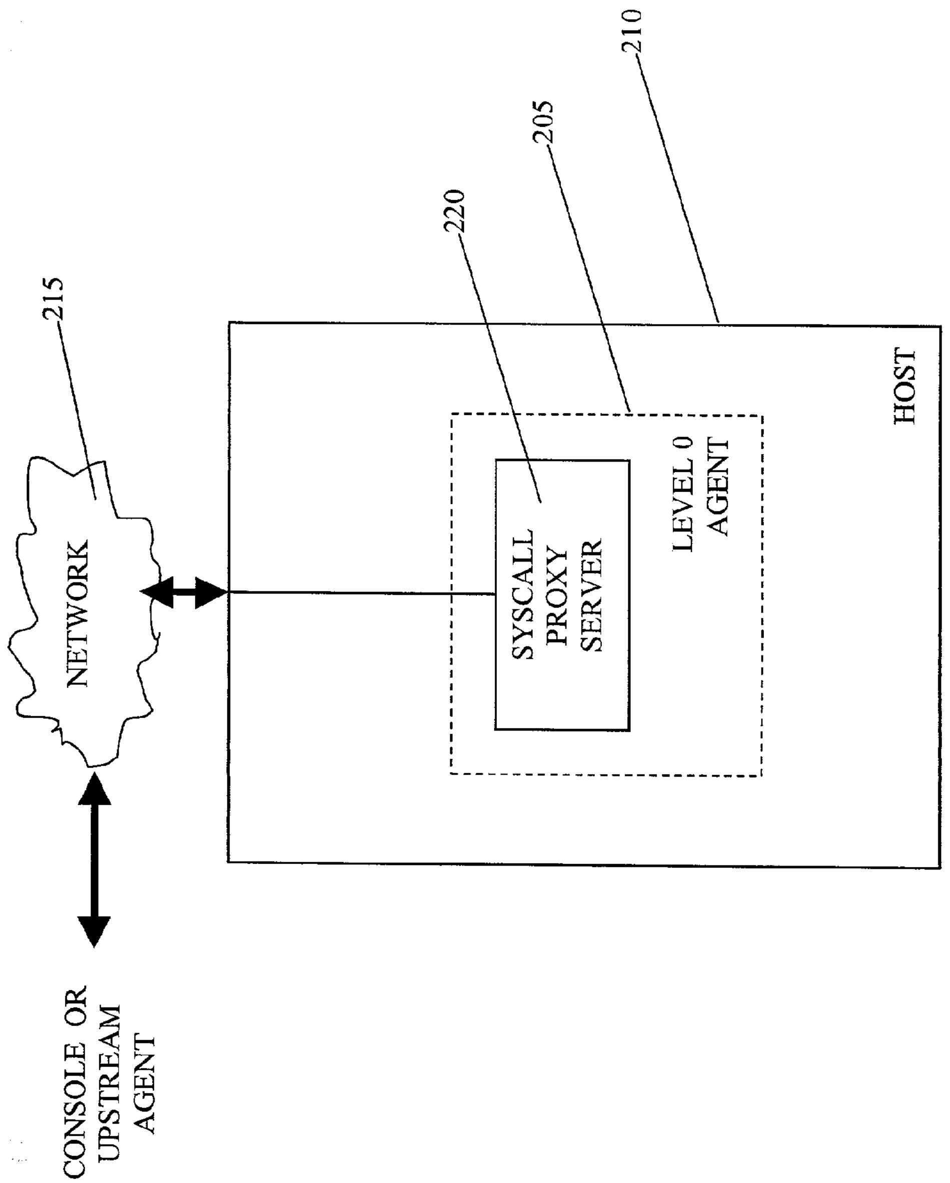

210
205
220
215
HOST
LEVEL 0
AGENT
SYSCALL
PROXY
SERVER
NETWORK
CONSOLE OR
UPSTREAM
AGENT

Fig. 5

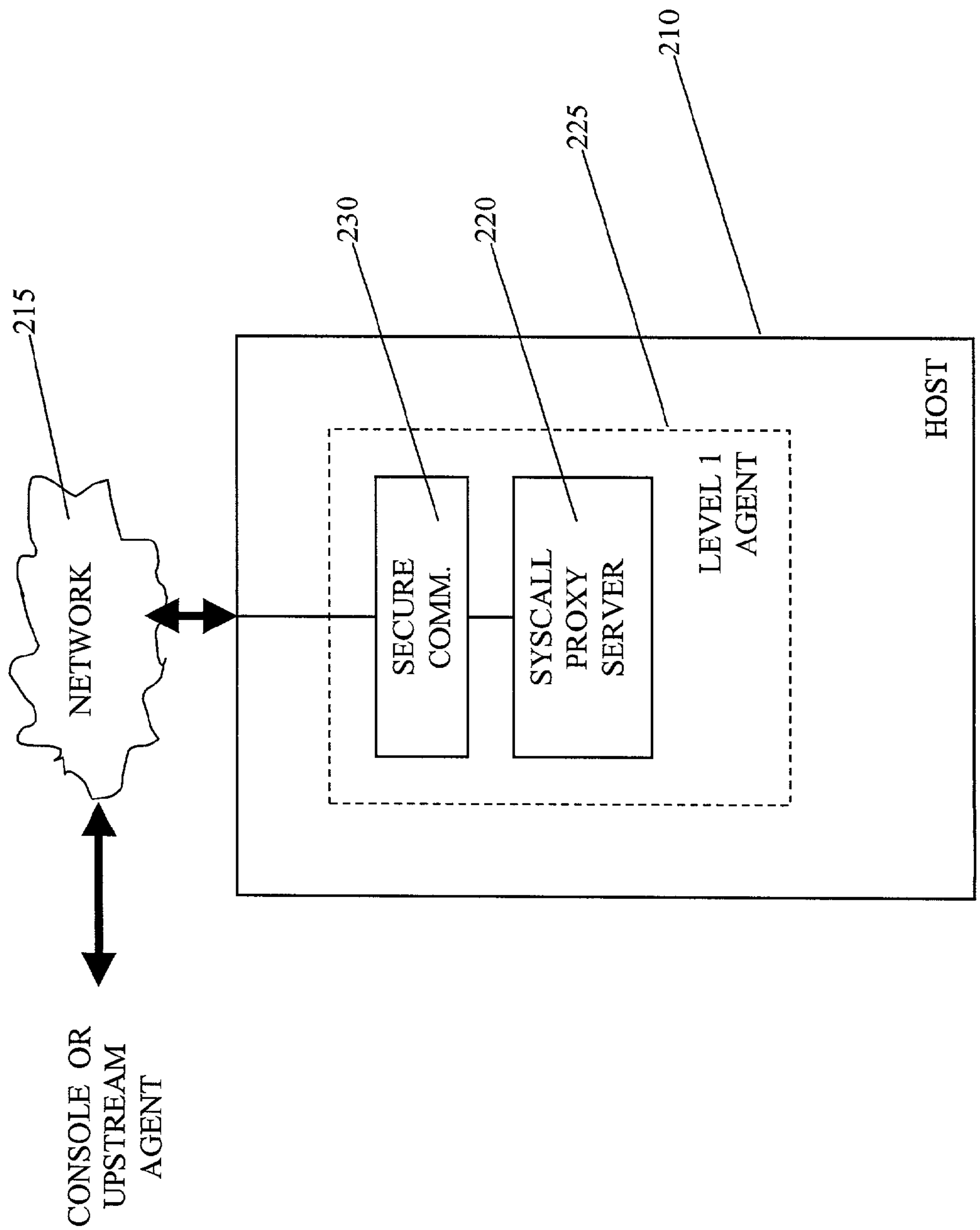

210
225
230
220
215
HOST
LEVEL 1 AGENT
SECURE COMM.
SYSCALL PROXY SERVER
NETWORK
CONSOLE OR UPSTREAM AGENT

Fig. 6

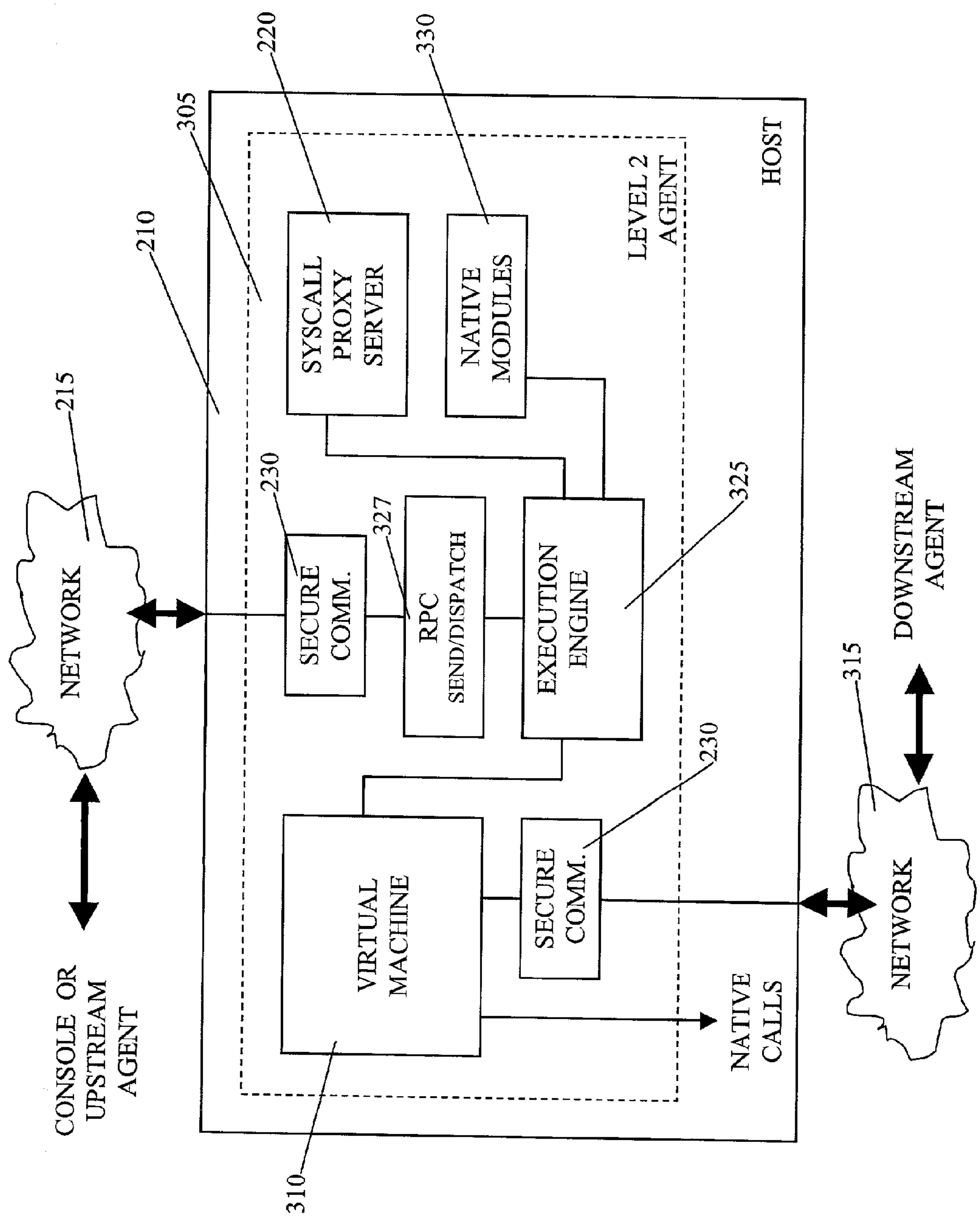

CONSOLE OR UPSTREAM AGENT
NETWORK
215
210
305
220
330
HOST
LEVEL 2 AGENT
SYSCALL PROXY SERVER
NATIVE MODULES
230
SECURE COMM.
327
RPC SEND/DISPATCH
EXECUTION ENGINE
325
VIRTUAL MACHINE
SECURE COMM.
230
NATIVE CALLS
310
NETWORK
315
DOWNSTREAM AGENT

Fig. 7

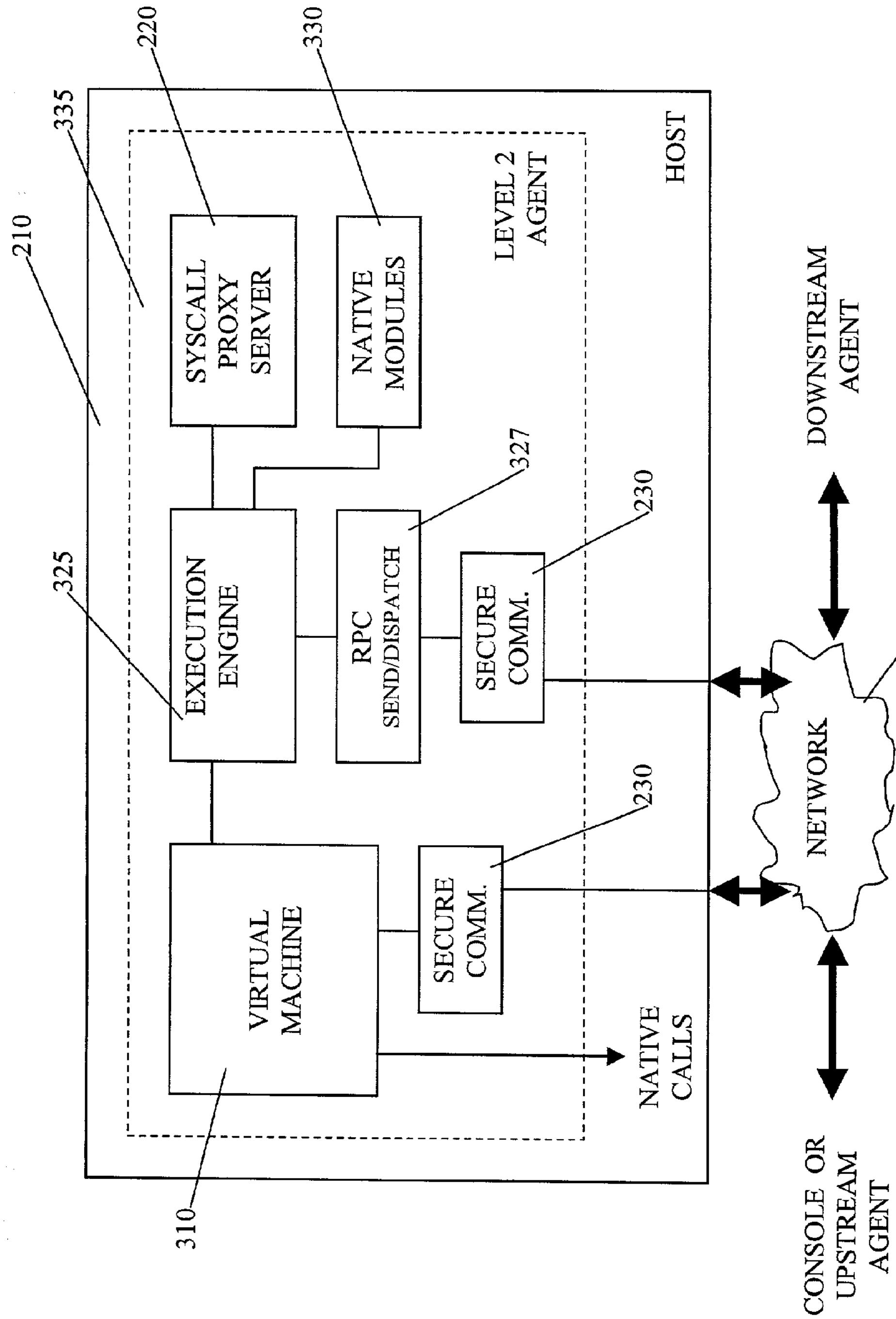

210
335
220
330
SYSCALL PROXY SERVER
NATIVE MODULES
LEVEL 2 AGENT
HOST
EXECUTION ENGINE
RPC SEND/DISPATCH
SECURE COMM.
325
327
230
VIRTUAL MACHINE
SECURE COMM.
230
NATIVE CALLS
310
DOWNSTREAM AGENT
NETWORK
320
CONSOLE OR UPSTREAM AGENT

Fig. 8

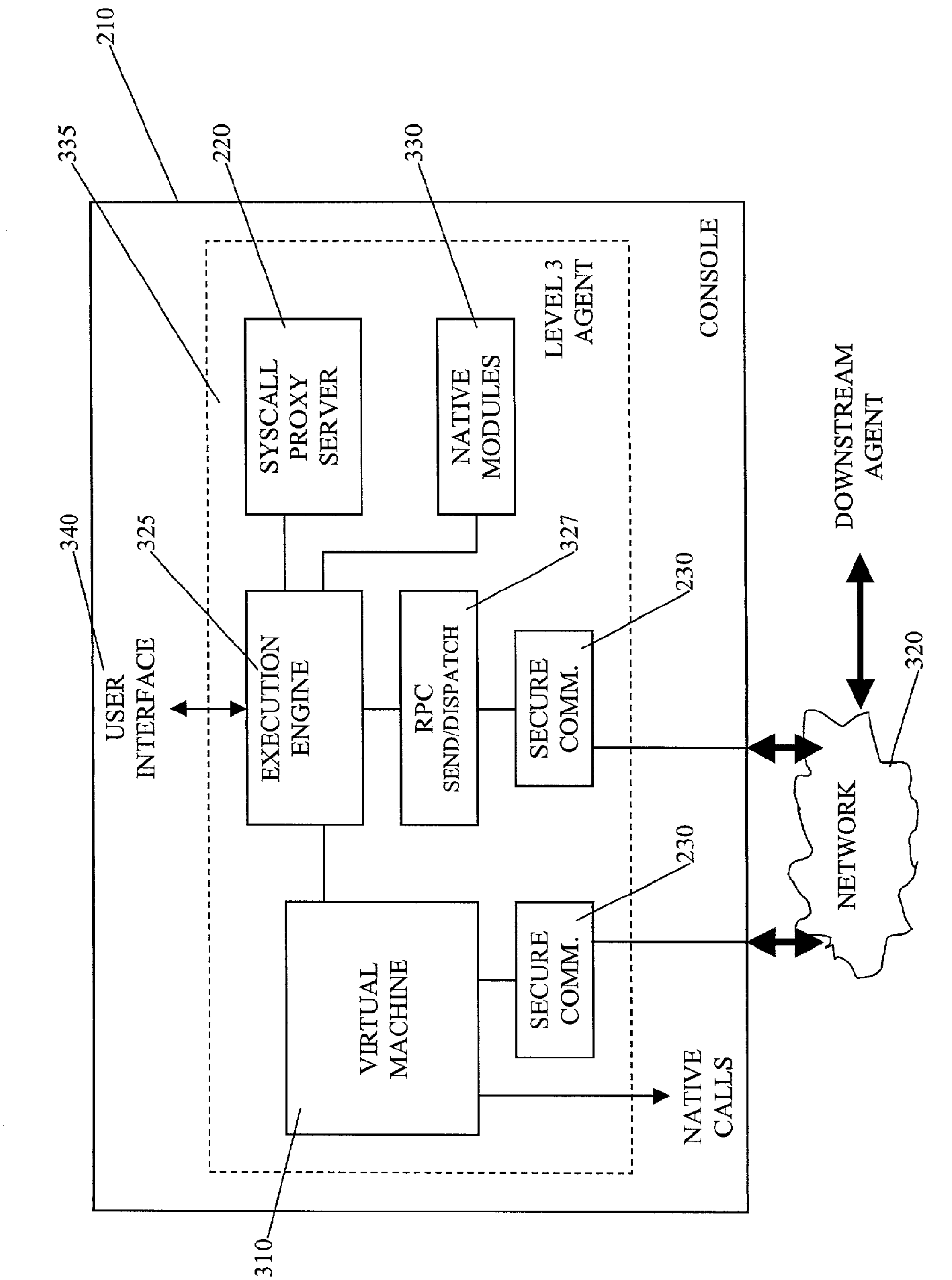

210
335
220
330
340
325
327
230
230
320
310
CONSOLE
LEVEL 3 AGENT
SYSCALL PROXY SERVER
NATIVE MODULES
USER INTERFACE
EXECUTION ENGINE
RPC SEND/DISPATCH
SECURE COMM.
SECURE COMM.
VIRTUAL MACHINE
NATIVE CALLS
NETWORK
DOWNSTREAM AGENT

Fig. 9

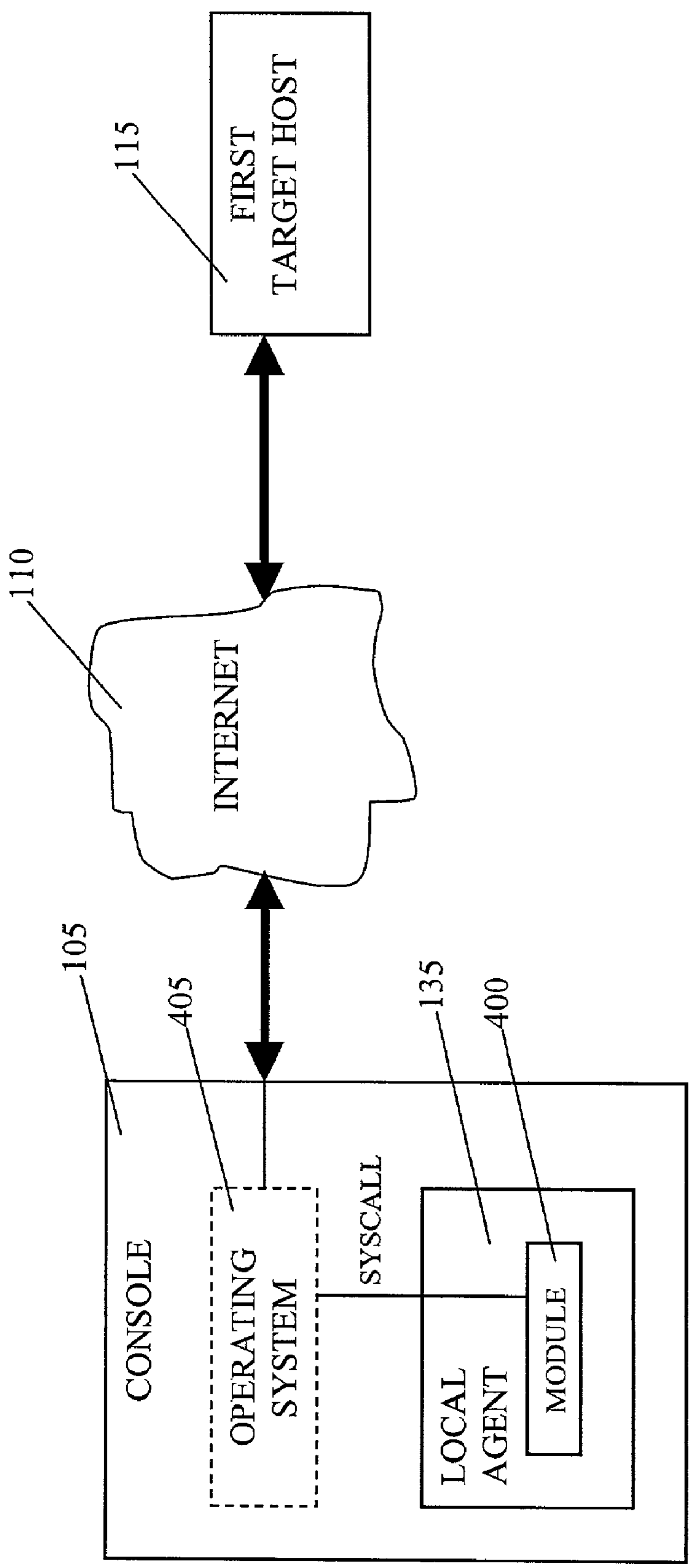

115
FIRST TARGET HOST
110
INTERNET
105
405
135
400
CONSOLE
OPERATING SYSTEM
SYSCALL
LOCAL AGENT
MODULE

Fig. 10

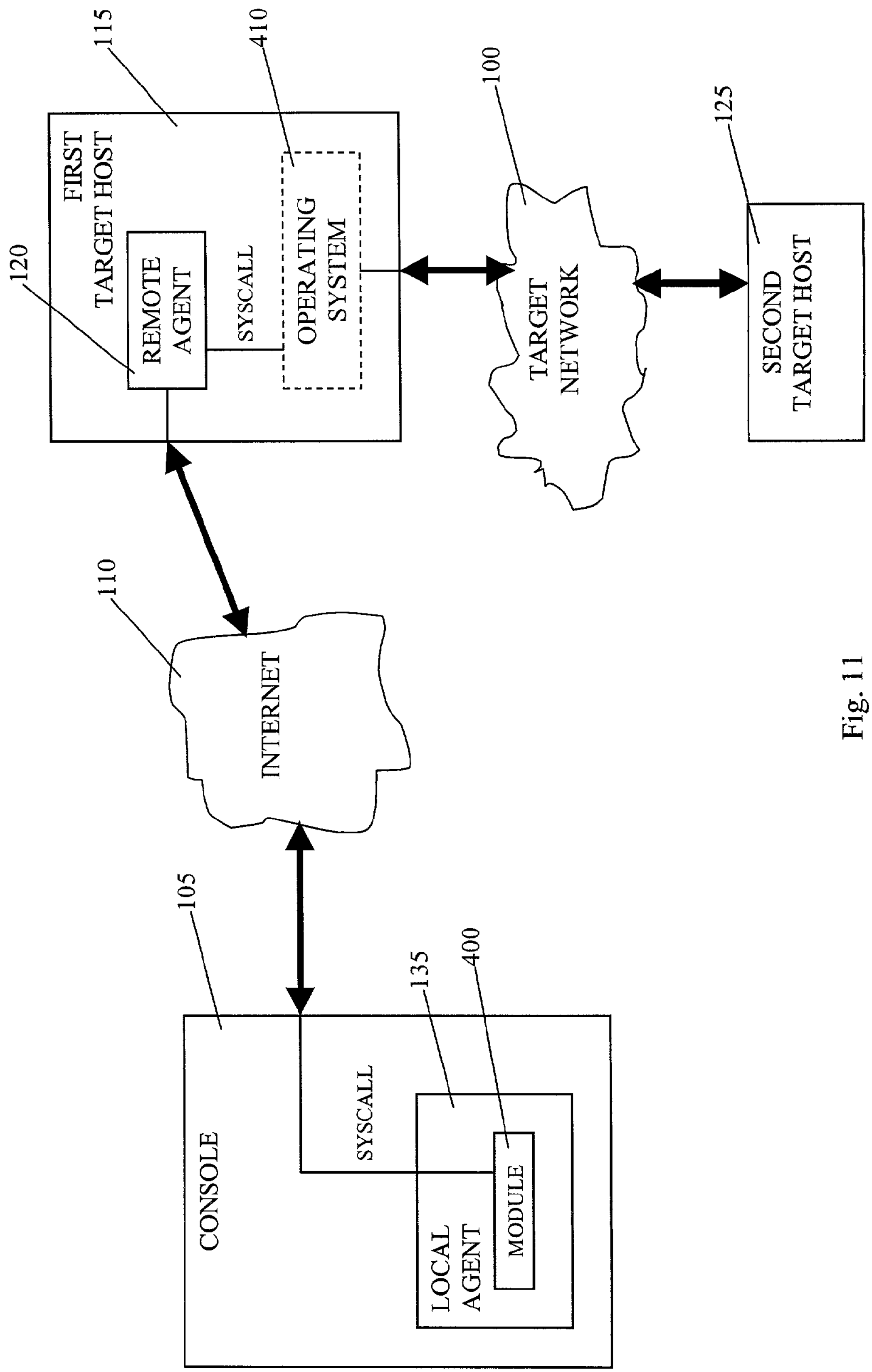

115
410
100
125
120
FIRST TARGET HOST
REMOTE AGENT
SYSCALL
OPERATING SYSTEM
TARGET NETWORK
SECOND TARGET HOST
110
INTERNET
105
135
400
CONSOLE
SYSCALL
LOCAL AGENT
MODULE

Fig. 11

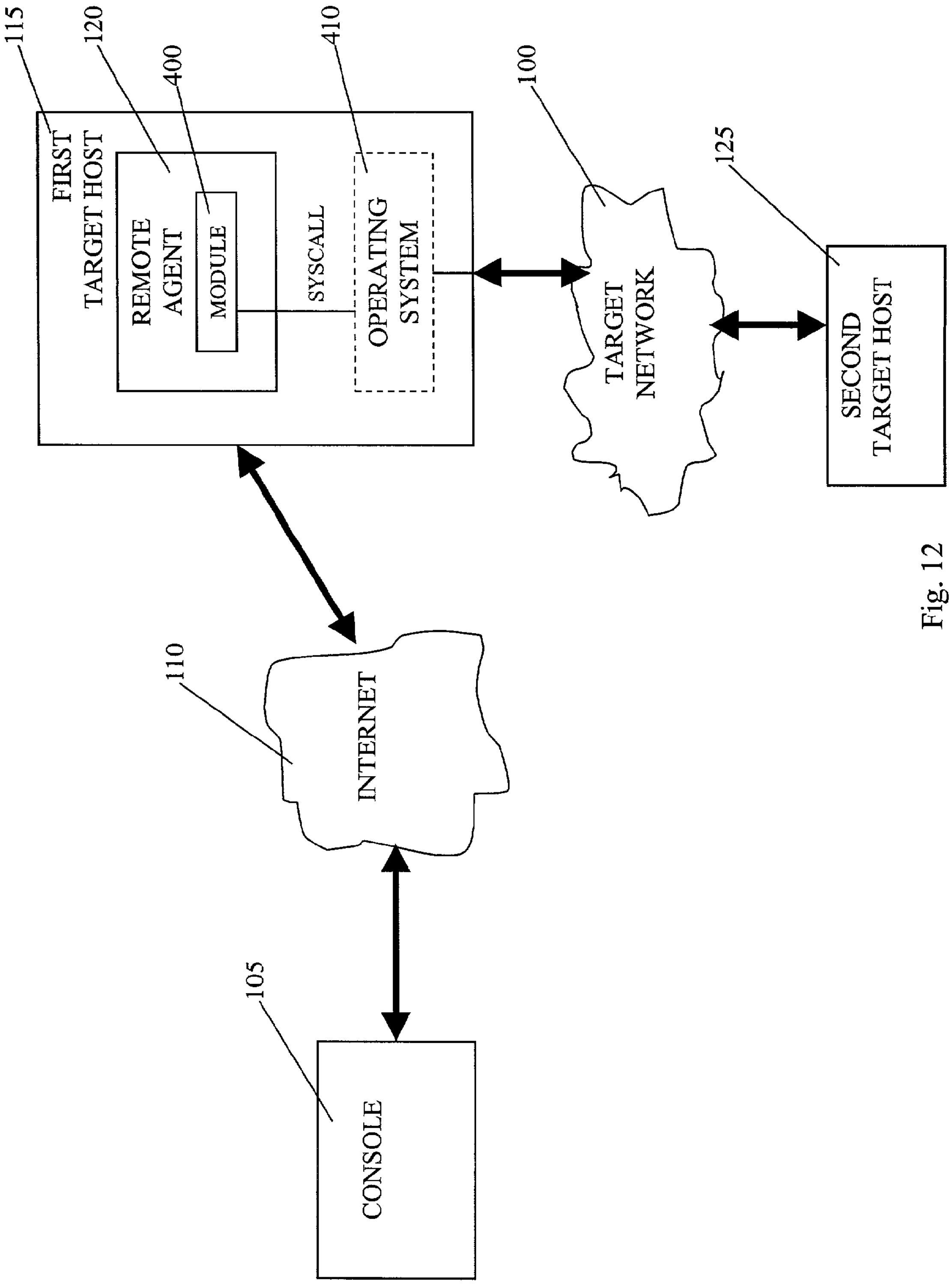

115
120
400
410
100
125
FIRST TARGET HOST
REMOTE AGENT
MODULE
SYSCALL
OPERATING SYSTEM
TARGET NETWORK
SECOND TARGET HOST
110
INTERNET
105
CONSOLE

Fig. 12

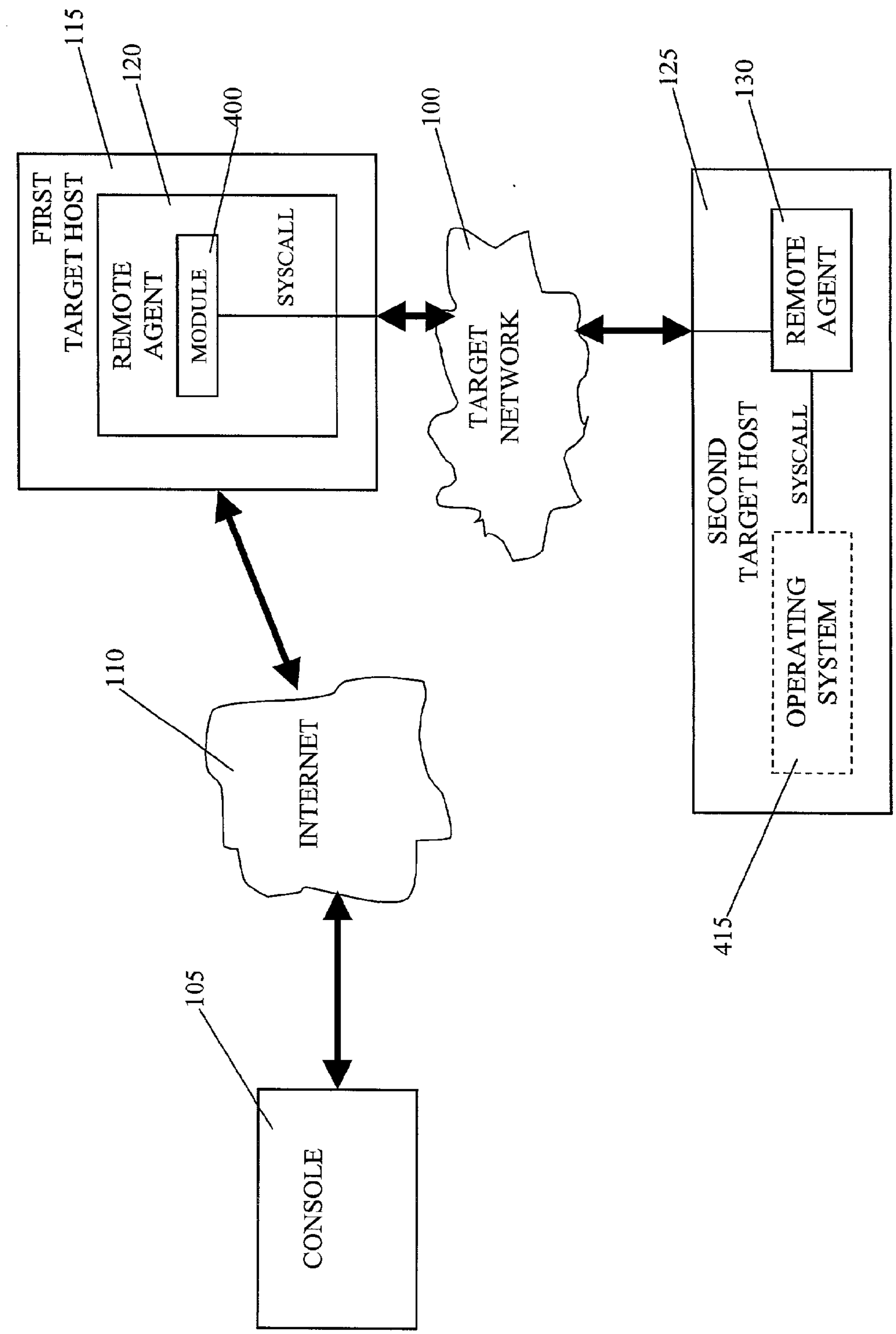

115
120
400
100
125
130
FIRST TARGET HOST
REMOTE AGENT
MODULE
SYSCALL
TARGET NETWORK
SECOND TARGET HOST
REMOTE AGENT
SYSCALL
OPERATING SYSTEM
110
INTERNET
415
105
CONSOLE

Fig. 13

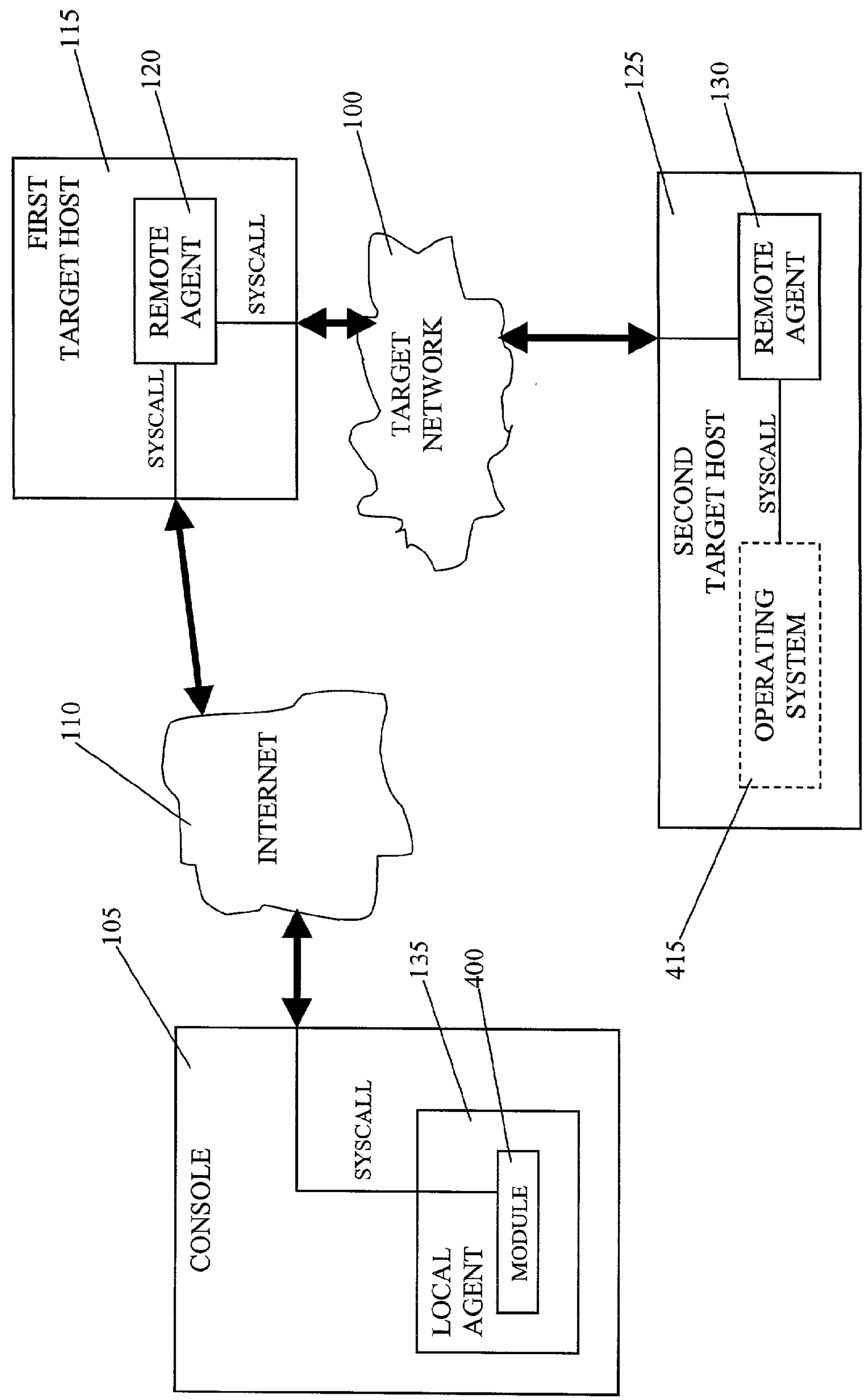

115
FIRST TARGET HOST
120
REMOTE AGENT
SYSCALL
SYSCALL
100
TARGET NETWORK
125
130
REMOTE AGENT
SECOND TARGET HOST
SYSCALL
OPERATING SYSTEM
415
110
INTERNET
105
CONSOLE
135
400
SYSCALL
LOCAL AGENT
MODULE

Fig. 14

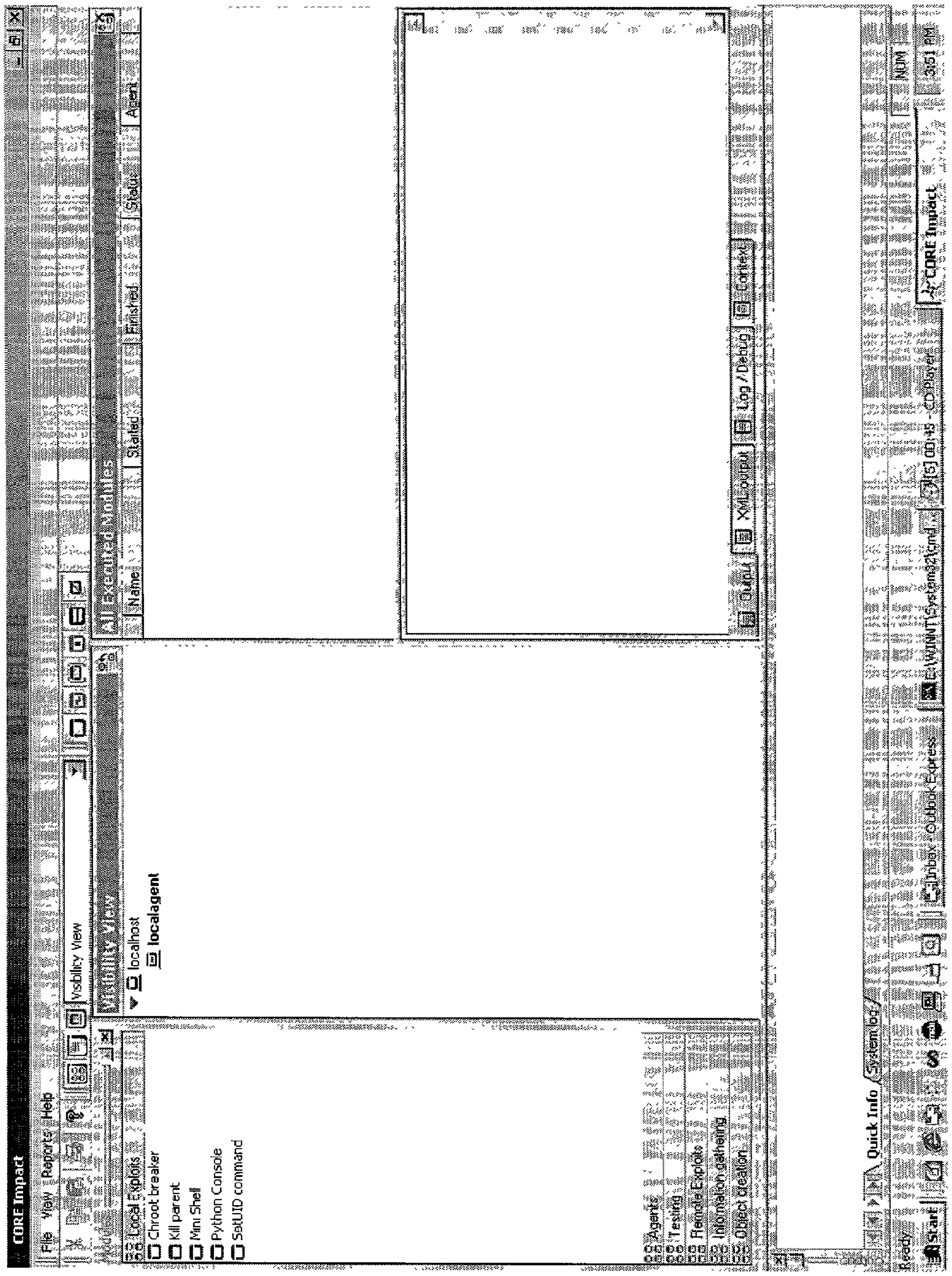

CORE Impact
File View Reports Help
Visibility View
Local Exploits
Chroot breaker
Kill parent
Mini Shell
Python Console
SetUID command
Agents
Testing
Remote Exploits
Information gathering
Object creation
localhost
localagent
All Executed Modules
Name
Started
Finished
Status
Agent
Output
XML output
Log / Debug
Context
Quick Info
System log
Ready
NUM
Start
CORE Impact
3:51 PM

Fig. 15

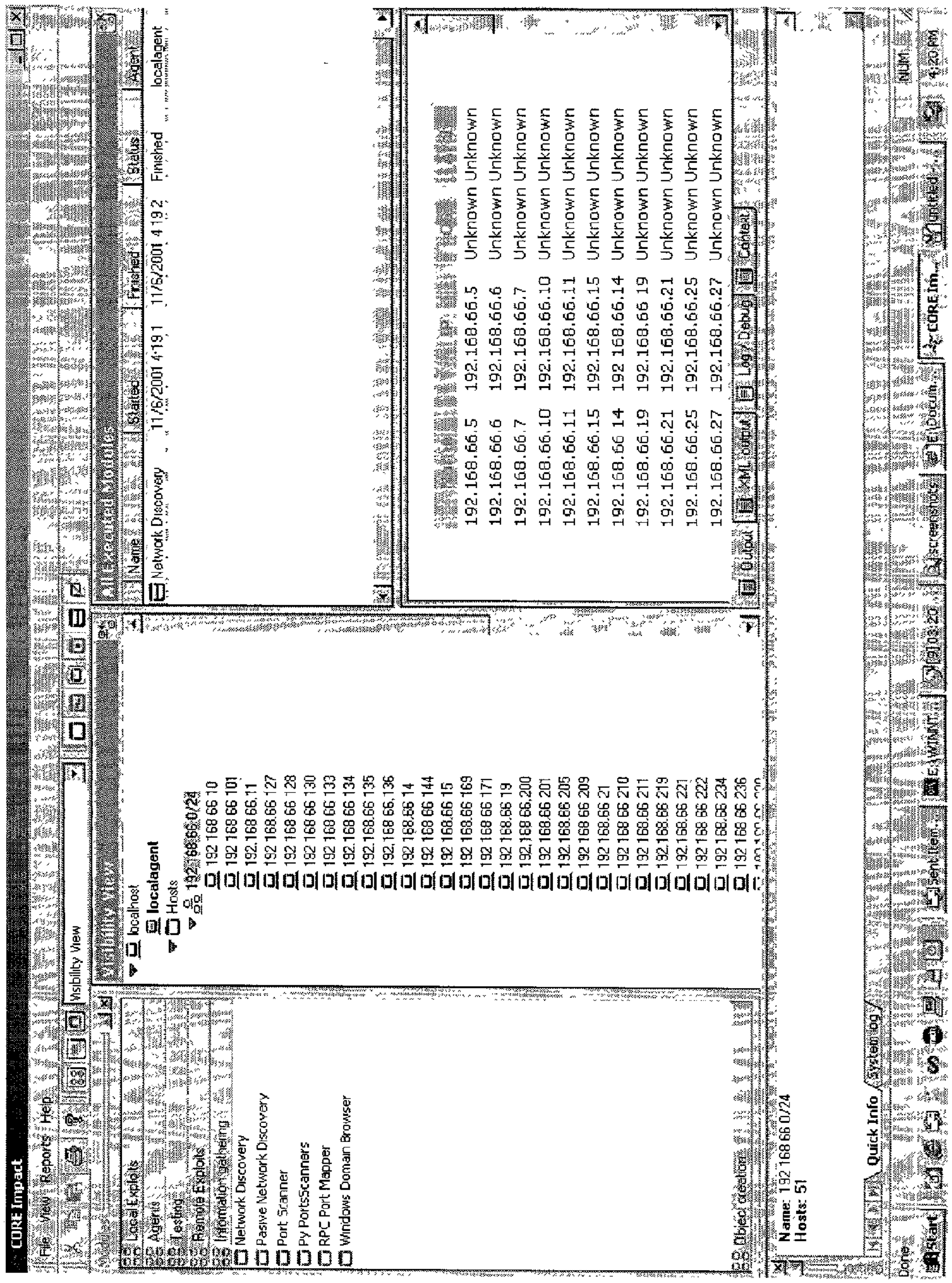


Fig. 16

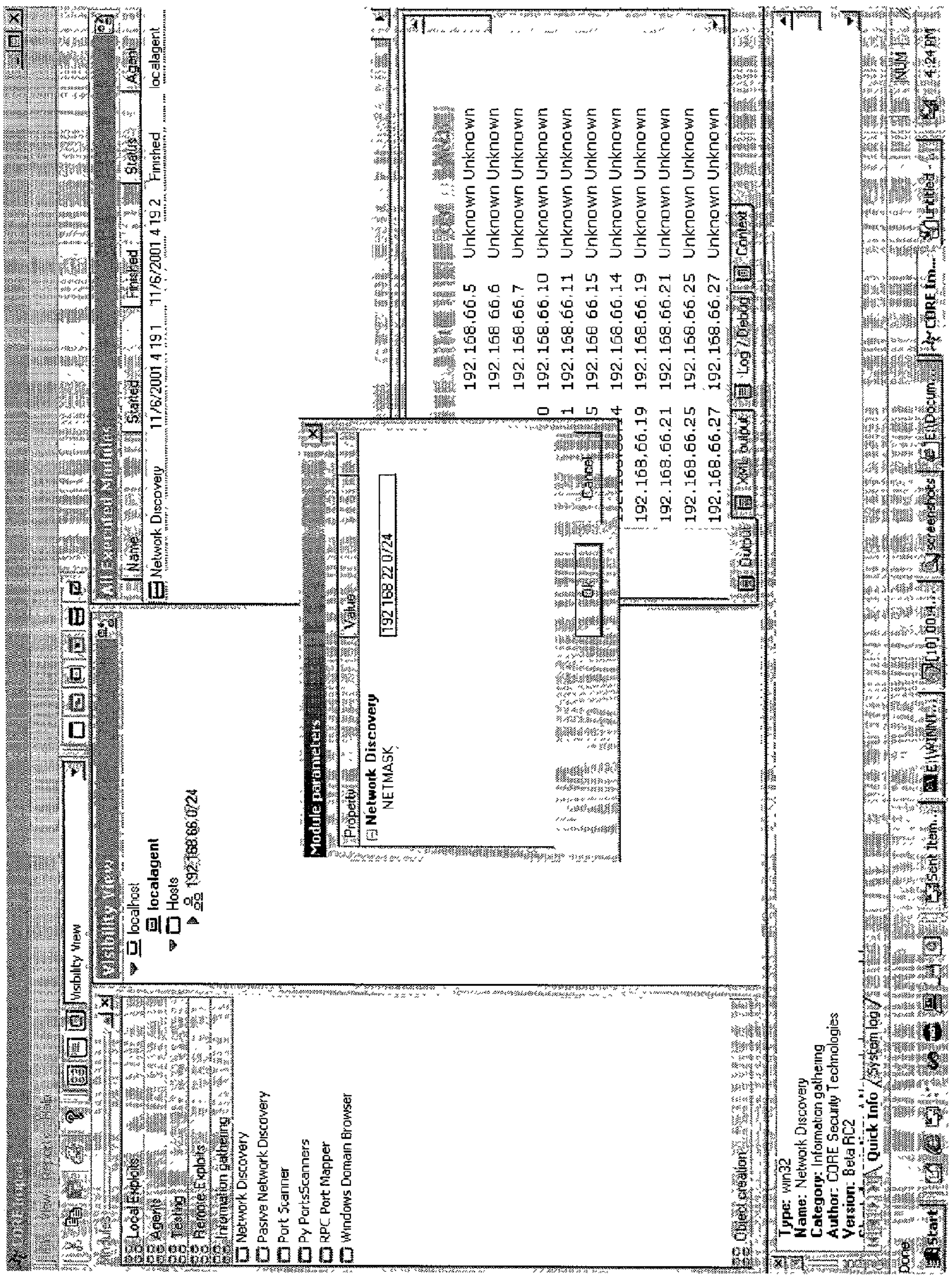


Fig. 17

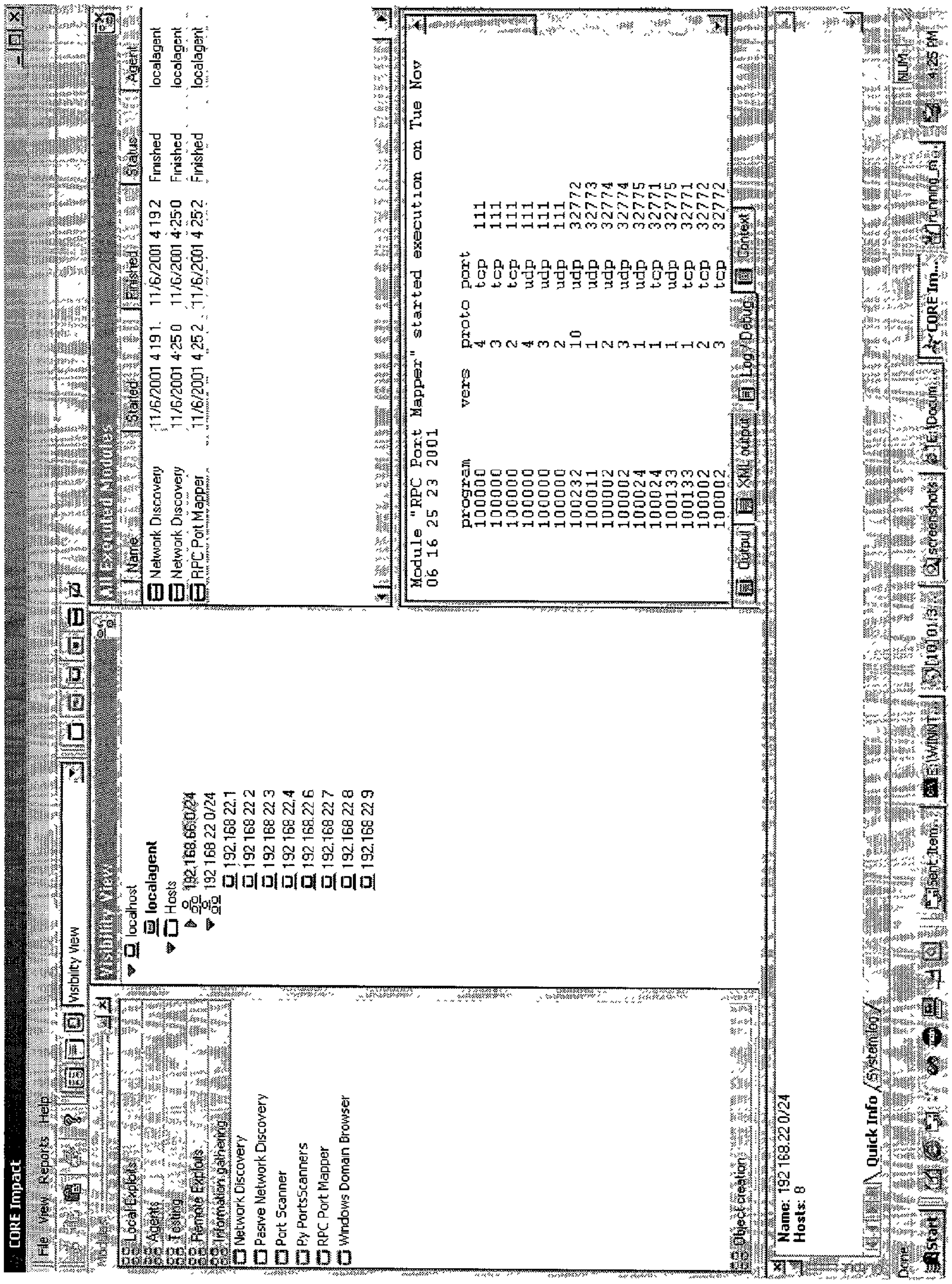

CORE Impact
Visibility View
Executed Modules
Network Discovery
Passive Network Discovery
Port Scanner
Py PortsScanners
RPC Port Mapper
Windows Domain Browser
localhost
localagent
Hosts
Module "RPC Port Mapper" started execution on Tue Nov 06 16 25 23 2001
program vers proto port
Quick Info
Name: 192.168.22.0/24
Hosts: 8

Fig. 18

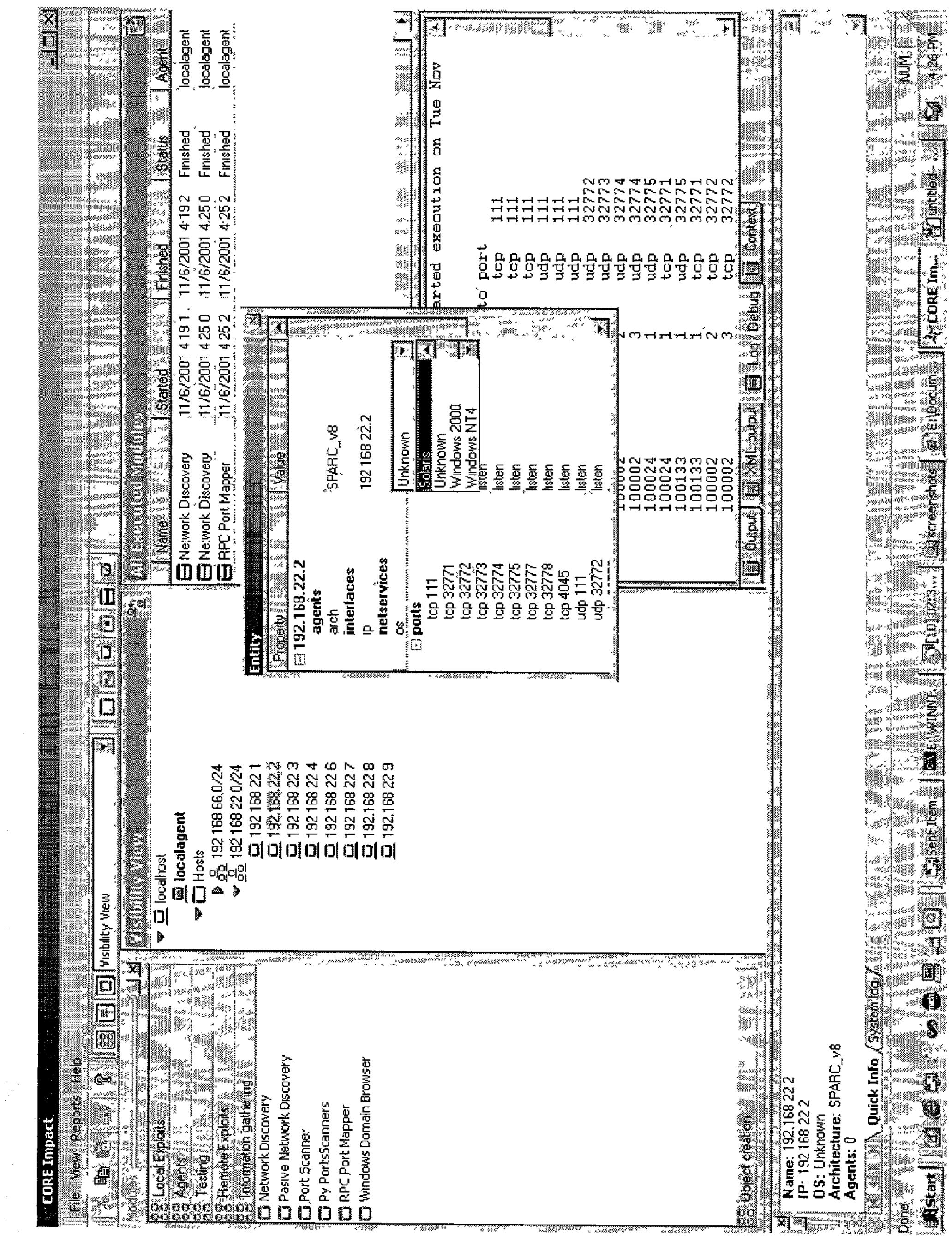


Fig. 19

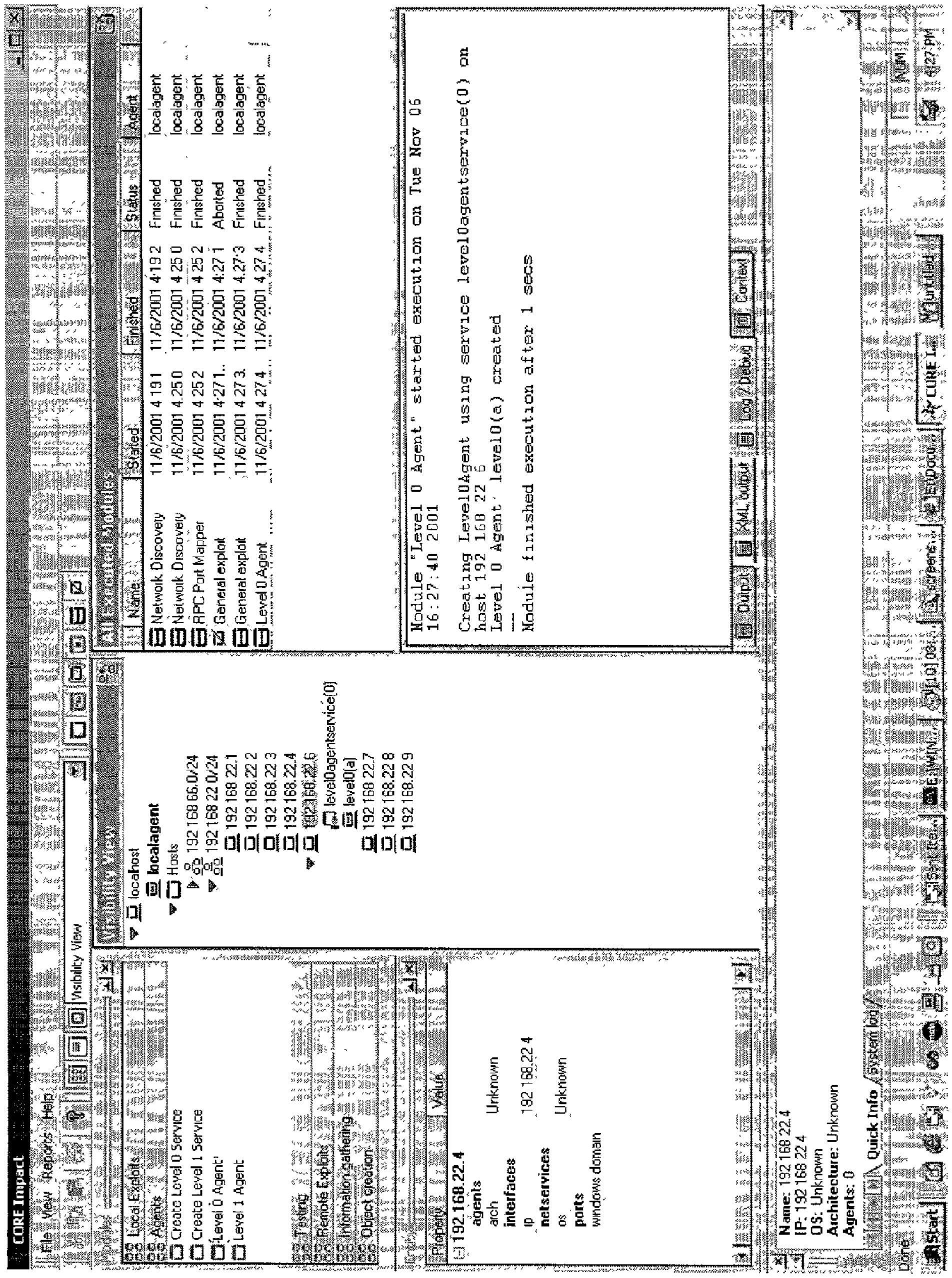

CORE Impact
All Executed Modules
Visibility View
Module "Level 0 Agent" started execution on Tue Nov 06 16:27:40 2001
Creating Level0Agent using service level0agentservice(0) on host 192 168 22 6
Level 0 Agent level0(a) created
Module finished execution after 1 secs

Fig. 20

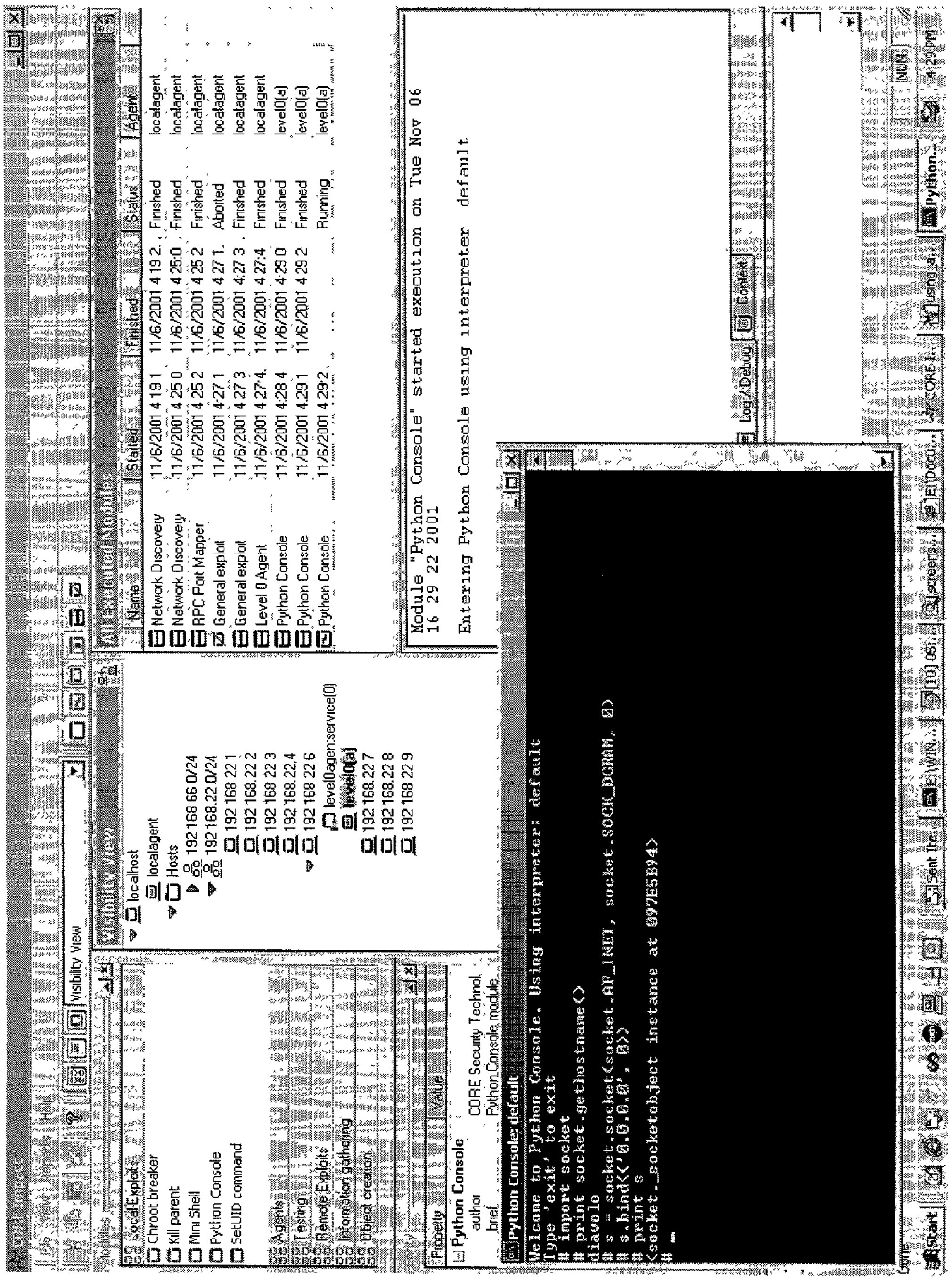

Visibility View
All Executed Modules
Python Console: default
Module "Python Console" started execution on Tue Nov 06 16 29 22 2001
Entering Python Console using interpreter default
Welcome to Python Console. Using interpreter: default
Type 'exit' to exit
import socket
print socket.gethostname()
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, 0)
print s

Fig. 21

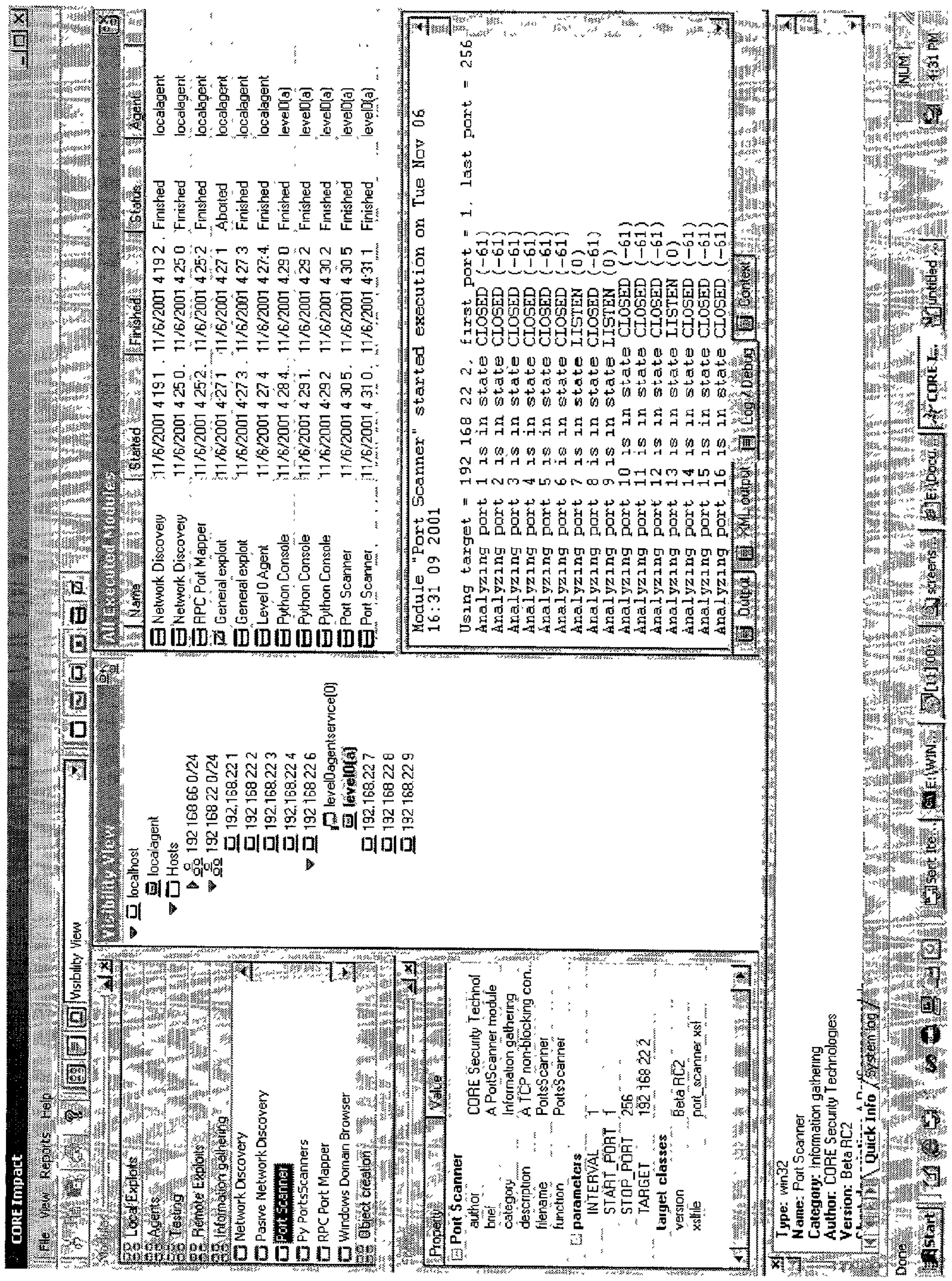


Fig. 22

AUTOMATED COMPUTER SYSTEM SECURITY COMPROMISE

[0001] This application claims the benefit of U.S. Provisional Application No. 60/304,270, filed Jul. 10, 2001, and U.S. Provisional Application No. 60/313,793, filed Aug. 20, 2001.

BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The present invention relates generally to analyzing computer system security by compromising the security in a systematic manner.

[0004] 2. Related Art

[0005] Computer systems that are connected to a computer network, such as the Internet, must employ security measures to prevent unauthorized users from accessing these systems. The security measures must be properly designed and implemented in order to prevent unauthorized access. However, it is difficult to evaluate the effectiveness of such security measures, particularly in view of the increasing sophistication of techniques used to gain unauthorized access to computer systems.

[0006] The effectiveness of the security measures of a computer system may be evaluated by performing a computer security audit in which various aspects of computer security are analyzed and evaluated. The security audit may include a penetration test, which is a process by which a security auditor attempts to gain unauthorized access to the computer system.

[0007] Conventionally, a penetration test is performed using a multitude of ad hoc methods and tools, rather than according to a formalized standard or procedure. A typical penetration test includes the following stages:

[0008] 1. Information gathering: The security auditor gathers technical details about the target system and information regarding the owner of the target system.

[0009] 2. Information analysis and planning: The auditor analyzes the information to plan an overall approach by which to perform the penetration testing. This tends to be a difficult and time consuming task and requires experienced and knowledgeable personnel having a highly specialized skill base.

[0010] 3. Vulnerability detection: The auditor searches the target system for security vulnerabilities based on the top-level plan developed in the information analysis and planning stage. Security vulnerabilities include, for example, system misconfigurations that enable an unauthorized user to gain access using a known series of steps.

[0011] The vulnerability search may be performed using an automated vulnerability scanner, which is a software package that determines whether certain known flaws may be used to gain unauthorized access to the target. Manual vulnerability scanning also may be performed to probe for common vulnerabilities that, for various reasons, may have been missed by an automated scanner. However, such vulnerability scanning techniques merely list the vulnerabilities, rather than actually attempt to exploit them.

[0012] The automated and manual vulnerability searches may be supplemented by research performed by the security auditor to determine previously unknown vulnerabilities. Such research typically is performed using a copy (also called a mirror) of the software application being probed and/or the associated hardware.

[0013] 4. Compromising and accessing the target system: The auditor attempts to compromise the target system based on the results of the vulnerability detection stage using publicly available or custom-developed programs.

[0014] Publicly available programs designed to exploit system vulnerabilities tend to be unreliable and require testing and customization before use. In general, exploiting detected vulnerabilities, regardless of the tools being used, requires experienced and knowledgeable personnel having a highly specialized skill base. In addition, a considerable laboratory infrastructure may be required to develop and test vulnerability exploitation tools, particularly when the target system employs a number of different operating system platforms.

[0015] 5. Analysis and reporting: This stage includes consolidating and presenting the information obtained during the previous stages and developing recommendations for remedying the security vulnerabilities identified during the penetration test. Manually maintaining a record of all of the actions taken and information gathered during testing is extremely time consuming and prone to error. Moreover, the preparation of complete and accurate records is subject to the discipline of the personnel conducting the test.

[0016] 6. Clean up: The compromising and accessing stage typically results in significant changes being made to the target system. In the clean up stage, the auditor returns the system to its original configuration. To perform a successful clean up, a detailed and exact list of all actions performed during testing must be maintained, yet there are only rudimentary tools available for maintaining such information.

[0017] In view of the shortcomings discussed above, there is a need for a system and method for performing an automated penetration test employing computer system compromise that takes an entirely fresh approach and overcomes the drawbacks of the conventional techniques.

SUMMARY OF THE INVENTION

[0018] The present invention generally provides a novel system, method, and computer code for performing automated penetration testing for analyzing computer system security by compromising the security in a systematic manner.

[0019] One aspect of the present invention provides a system, method, and computer code for performing penetration testing of a target computer network by installing a remote agent in the target computer network. A local agent is provided in a console and configured to receive and execute commands. A user interface provided in the console and configured to send commands to and receive information from the local agent, process the information, and present the processed information. A database is configured to store the information received from the local agent. A

network interface is connected to the local agent and configured to communicate via a network with the remote agent installed in the target computer network. Security vulnerability exploitation modules are provided for execution by the local agent and/or the remote agent.

[0020] Embodiments of this aspect may include one or more of the following features. The user interface may enable a user to select one of the modules and initiate execution of the selected module on either the local agent or the remote agent. The user interface may provide a graphical representation of the target computer network.

[0021] Another aspect of the present invention provides an agent for use in a system for performing penetration testing of a target computer network. The agent includes a proxy server configured to receive and execute system calls received via a network and a virtual machine configured to execute scripting language instructions received via the network.

[0022] Embodiments of this aspect may include one or more of the following features. The agent may include an execution engine configured to control the proxy server and the virtual machine. The system calls and the scripting language instructions may be routed to the proxy server and the virtual machine, respectively, by the execution engine. The agent may include a remote procedure call module configured to receive commands from the network formatted in a remote procedure call protocol and pass the commands to the execution engine.

[0023] Another aspect of the present invention provides an agent including a proxy server configured to receive and execute system calls received via a network and a virtual machine configured to execute scripting language instructions received via the network. The agent further includes a secure communication module configured to provide secure communication between the virtual machine and the network. An execution engine is configured to control the proxy server and the virtual machine. The system calls and the scripting language instructions are routed to the proxy server and the virtual machine, respectively, by the execution engine. A remote procedure call module is configured to receive commands via the network formatted in a remote procedure call protocol and pass the commands to the execution engine. A second secure communication module is configured to provide secure communication between the remote procedure call module and the network.

[0024] Another aspect of the present invention provides a system, method and computer code for performing penetration testing of a target network in which a first module is executed in a console having a user interface. The first module is configured to exploit a security vulnerability in a first target host of the target network. A first remote agent is installed in the first target host. The first remote agent is configured to communicate with the console and a second remote agent. A second module is executed in the first remote agent. The second module is configured to exploit a security vulnerability in a second target host of the target network.

[0025] Embodiments of this aspect may include the feature of installing a second remote agent in the second target host of the target network, the second remote agent being configured to communicate with the first remote agent.

[0026] Another aspect of the present invention provides a system, method, and computer code for performing penetration testing of a target network. A first module is executed to exploit a security vulnerability of a first target host of the target network. A first remote agent is installed in the first target host as a result of exploiting the security vulnerability of the first target host. A system call is sent to the first remote agent via a network. The system call is executed in the first target host using a proxycall server of the first remote agent to exploit a security vulnerability of a second target host.

[0027] In another aspect, a first module is executed to exploit a security vulnerability of a first target host of the target network. A first remote agent is installed in the first target host as a result of exploiting the security vulnerability of the first target host. A second module that generates a system call is executed in the first remote agent. The system call is executed in the first target host to exploit a security vulnerability of a second target host.

[0028] In another aspect, a first module is executed to exploit a security vulnerability of a first target host of the target network. A first remote agent is installed in the first target host as a result of exploiting the security vulnerability of the first target host. A second module is executed in the first remote agent that generates a system call. A second remote agent is installed in the second target host as a result of exploiting a security vulnerability of the second target host. The system call generated by the second module is sent to the second remote agent via a network. The system call is executed in the second target host using a proxycall server of the second remote agent.

[0029] In another aspect, a first module is executed to exploit a security vulnerability of a first target host of the target network. A first remote agent is installed in the first target host as a result of exploiting the security vulnerability of the first target host. A second remote agent is installed in the second target host as a result of exploiting a security vulnerability of the second target host. A system call is sent to the first remote agent and is sent from the first remote agent to the second remote agent. The system call is executed in the second target host using a proxycall server of the second remote agent.

[0030] In another aspect, a first remote agent is installed in the first target host. The first remote agent has a proxy server configured to receive and execute system calls. A system call received via a network is executed in the first remote agent. A second remote agent is installed in the first target host. The second remote agent has a proxy server configured to receive and execute system calls and a virtual machine configured to execute scripting language instructions. A scripting language instruction or a system call received via the network is executed in the second remote agent.

[0031] These and other objects, features and advantages will be apparent from the following description of the preferred embodiments of the present invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0032] The present invention will be more readily understood from a detailed description of the preferred embodiments taken in conjunction with the following figures.

[0033] **FIG. 1** is a block diagram of a system for performing automated penetration testing of a target network in accordance with the present invention.

[0034] **FIG. 2** is a block diagram of a console for automated penetration testing connected through the Internet to a first target host.

[0035] **FIG. 3** is a block diagram of the console connected through the Internet to a first target host, which is connected through a target network to a second target host.

[0036] **FIG. 4** is a block diagram showing components of the console.

[0037] **FIG. 5** is a block diagram of a level 0 agent.

[0038] **FIG. 6** is a block diagram of a level 1 agent.

[0039] **FIG. 7** is a block diagram of a level 2 agent connected to upstream and downstream agents by two different networks.

[0040] **FIG. 8** is a block diagram of a level 2 agent connected to upstream and downstream agents by a single network.

[0041] **FIG. 9** is a block diagram of a level 3 agent.

[0042] **FIG. 10** is a block diagram of a configuration in which a module is executed by the local agent in the console and system calls are executed in the operating system of the console.

[0043] **FIG. 11** is a block diagram of a configuration in which a module is executed by the local agent in the console and system calls are executed in the operating system of the first target host.

[0044] **FIG. 12** is a block diagram of a configuration in which a module is executed by the remote agent in the first target host and system calls are executed in the operating system of the first target host.

[0045] **FIG. 13** is a block diagram of a configuration in which a module is executed by the remote agent in the first target host and system calls are executed in the operating system of the second target host.

[0046] **FIG. 14** is a block diagram of a configuration in which a module is executed by the local agent in the console and system calls are executed in the operating system of the second target host.

[0047] **FIG. 15** is a basic graphical user interface display screen presented by the user interface of the console.

[0048] **FIG. 16** is a display screen following the running a Network Discovery Module.

[0049] **FIG. 17** is a display screen following initiation of the Network Discovery module for a second time.

[0050] **FIG. 18** is a display screen showing the results of the second running of the Network Discovery module and the running of the Remote Procedure Call (RPC) Mapper module.

[0051] **FIG. 19** is a display screen in which the entities in the model of the target network can be examined and modified using an entity editor.

[0052] **FIG. 20** is a display screen following two executions of the General Exploit module and the installation of a level 0 agent in a target host.

[0053] **FIG. 21** is a display screen showing a window in which a Python console is running using the level 0 agent as a source.

[0054] **FIG. 22** is a display screen following the execution of a port scanning module using the level 0 agent as a source.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0055] According to the present invention, as shown in **FIG. 1**, an automated penetration test is performed to identify, analyze, exploit, and document security vulnerabilities in a target network **100**. The penetration test is executed by a console **105** that may be, for example, a personal computer running Microsoft Windows 2000 Professional, Server, or Advanced Server operating systems. The target network **100** may be connected to a network, such as for example the Internet **110**. In the case of such example, the console also would be connected to the Internet **110** and would gain access to the target network **100** through the Internet **110**.

[0056] The target network **100** has a first target host **115**, e.g., a firewall. The firewall is a security device that typically is the only host in the target network that is connected directly to the Internet. It handles Internet traffic to and from the target network and serves to protect the target network from unauthorized access. The target network **100** has a number of other hosts connected to it-all of which could be the eventual targets of the penetration test.

[0057] The console **105**, shown in **FIG. 2**, compromises the security measures protecting the first target host **115** by executing a series of modules. The modules may be selected and initiated by the user. Alternatively, the console may execute a predetermined sequence of modules or may determine a sequence of modules to be executed based on the information gathered during the penetration testing.

[0058] In the initial stage, typically, modules are executed to gather information about the first target host **115**. For example, the console **105** may execute a port scanner that analyzes the ports of the first target host **115** and determines all of the services that are being run, such as an Internet web server, an email server, a finger service, etc. Further information might be acquired by running modules designed to exploit the services identified by the port scanner. For example, if the first target host **115** is running a finger service, then that service will be targeted by a module to determine software version, user names, etc. As a further example of an information gathering module, a network discovery module may be used to determine the number of hosts in the target network **100** and the Internet Protocol (IP) address of each host.

[0059] Following execution of the information gathering modules, the console executes modules, called exploits, to exploit security vulnerabilities in the first target host **115** based on the information that has been retrieved. For example, information may be obtained regarding a firewall operating system being run on the first target host **115**, such as the software brand and revision number. Based on this information, the console **105** executes an exploit that has been written to take advantage of security vulnerabilities for that particular firewall.

[0060] Once a service running on the first target host **115** has been compromised, the console **105** installs a remote

agent **120** on the first target host. The remote agent **120** is a program that operates on the first target host **115** to perform a number of functions, such as receiving and executing control commands and modules from the console **105** and sending back information to the console **105**.

[**0061**] As shown in **FIG. 3**, once the remote agent **120** has been installed on the first target host **115**, it is used by the console **105** to gain access to the target network **100** and compromise the security of the other hosts that make up the target network **100**, such as the second target host **125**. For example, the remote agent **120** on the first target host **115** executes exploits, such as those discussed above, or system calls received from the console **105** to gather information and exploit other security vulnerabilities in the target network **100**. To hosts connected to the target network **100**, such commands or queries appear to originate from the first target host **115** and therefore may be more readily accepted. This is particularly true for networks that employ hierarchies of trust, which means that commands from known, i.e., trusted, sources are subject to less stringent security measures than commands from unknown sources.

[**0062**] Once the security of the second target host **125** has been compromised, the remote agent **120** in the first target host **115** installs a remote agent **130** in the second target host **125**. Each of the installed agents **120** and **130** sends and receives modules, commands, and data from other installed agents, which is referred to as chaining. For example, the agent **130** in the second target host **125** receives modules and commands from the agent **120** in the first target host **115**, which, in turn, receives the modules and commands from the console. The agent **130** in the second target host **125** also sends data back to the agent **120** in the first target host **115**, which, in turn, sends the data back to the console **105**.

[**0063**] The agent **130** in the second target host **125** executes the modules received from the upstream agents to gather information from and exploit security vulnerabilities in a third target host, in a manner similar to that discussed above. Once the security measures have been compromised, the agent in the second target host installs an agent in the third target host. The penetration of the target network may continue in this manner until all of the target hosts have been compromised or until the final target of the penetration testing has been compromised.

[**0064**] The term "exploiting security vulnerabilities", as used herein, is a broad concept that includes any means for gaining access to and/or obtaining information from a target host. The concept includes, without limitation, the execution of modules that are designed to take advantage of specific security vulnerabilities that have been identified in a target host. For example, the first target host **115** may have been misconfigured by the owner in a manner that is detectable and allows installation of a remote agent **120**. This concept also includes, without limitation, the execution of information gathering modules, such as port scanners and network discovery modules. This concept further includes, without limitation, the exploitation of security vulnerabilities of a target host that result from the compromise of other target hosts. For example, once a remote agent **120** has been installed in the first target host **115**, it maybe possible to gather information and install remote agents on other target hosts due to hierarchies of trust within the target network. This concept further includes, without limitation, obtaining access to a target host by virtue of a lack of security features or measures.

[**0065**] As shown in **FIG. 4, a** local agent **135** installed in the console **105** communicates with and controls the remote agents **120** and **130** through a network interface **140**, such as a network interface card. The console **105** provides a module repository **145** to store modules that perform various functions related to system penetration and compromise. During the penetration test, these modules are sent to the local agent **135** in the console **105** or a remote agent **120** or **130** in a target host **115** or **125** to be executed. For example, a module that performs packet sniffing may be sent to and executed on the remote agent **120** in order to monitor data packets passing through the first target host **115**.

[**0066**] The console **105** also provides a user interface **150** that enables the user to control and monitor the performance of the penetration test. The user interface **150** may be a graphical user interface that displays a representation of the target network **100**. The user interface **150** also reports the results of the activities performed by the agents installed in the target network.

[**0067**] A database **155** in the console **105** stores information sent back by remote agents **120** and **130**. The database **155** could be a standard flat file, for example a text file, although a more complex structure, such as a relational database, also may be used. In addition, data may be exported to an external relational database. Each structured element of information is represented as an object in the database **155**. The database **155** can accommodate multiple distributed instances of the same object that can be independently updated. The database **155** synchronizes the multiple instances of the objects to allow revision of the instances with the most recent data.

[**0068**] The information that is stored as objects in the console database **155** includes an activity log of all actions performed by the remote agents **120** and **130**, such as executing modules, port scanning the host, passing modules to downstream remote agents, and installing remote agents in other hosts. The information also includes configuration data relating to the deployment of the remote agents, such as identification of the hosts in which the remote agents are installed and the possible communication channels that might be used to connect to them. The information also includes all of the data generated by execution of commands and modules by the remote agents, such as known hosts in the target network, operating systems, open ports, user accounts, cracked passwords, etc. The remote agents may store a subset of the database to provide caching of module data and to make such data available to downstream modules. The cached data is eventually sent back to the console database.

[**0069**] Another type of object that can be managed by the database **155** is referred to as an entity, which is an object that represents a physical component in the target network **100**, such as a host. Entities contain information similar to that discussed above, but also may have the capability to perform certain functions. For example, an entity may be capable of serializing and deserializing itself to and from XML markup language for transfer in and out of the database. This allows information produced by modules executed by remote agents to be shared between agents and with the console.

[0070] The data stored in the console database **155** is used for a number of purposes, such as to analyze the security vulnerabilities of the target network **100** and plan the exploitation of these vulnerabilities to penetrate further into the target network **100**. The data also allows the console to reverse any changes made to the target network **100** during the penetration testing. In addition, the data allows the client for whom the penetration testing is performed to have a complete picture of actions performed during the testing. For example, the data may be exported into a relational database, as mentioned above, or may be used to generate reports using XML.

[0071] In general terms, an agent, such as the local and remote agents discussed above, is a program that is controlled by or used as a proxy by another program. The controlling program may be resident on the same host as the agent, as in the case of the user interface that controls the local agent on the console. Alternatively, the controlling program may be on a separate host, as in the case of the local agent on the console that controls the remote agent on the first target host. Hence, in this example, the local agent in the console serves both as an agent and as a controlling program.

[0072] **FIG. 5** shows an example of a basic agent, which is referred to as a level 0 agent **205**, running on a host **210** that is connected to a network **215**. The level 0 agent **205** communicates through the network **215** to a controlling program running on an upstream agent, such as the local agent **135** in the console **105** or an intermediate agent positioned between the console and the level 0 agent. The level 0 agent **205** provides a syscall proxy server **220** that enables it to act as a proxy for the controlling program. This configuration is referred to as a system call proxy or proxy-call configuration.

[0073] For example, referring again to **FIG. 4, a** level 0 agent **205** may be installed as the remote agent **120** on the first target host **115** and may be controlled by a module executed by the local agent **135** in the console **105** such as, for example, a remote exploit. The module includes commands that result in system calls ("syscalls"), which are instructions that access the operating system of the host. Rather than executing the syscall on the console **105**, the syscalls are sent by a proxy client in the local agent **135** to the syscall proxy server **220** in the remote agent **120**. The syscall proxy server **220** executes the syscalls on the first target host **115**. Thus, syscalls are executed on the host of the remote agent **120**, e.g., the first target host **115**, rather than the host of the local agent **135**, e.g., the console **105**.

[0074] By acting as a syscall proxy, the remote agent **120** allows the local agent **135** to execute commands as if the local agent **135** were resident on the first target host **115**. The syscall proxy configuration allows large, complex programs, such as the automated penetration testing program resident on the console **105**, to execute system calls on a target host without actually being resident on the target host. Only the relatively small agent needs to be installed on the target host.

[0075] The level 0 agent **205** is a relatively simple program and may be about 100 bytes in size or smaller. Due to its small size, the level 0 agent **205** is typically the first type of agent to be installed in the target host during penetration testing. It may be used to gather information and/or exploit vulnerabilities in the target host to enable further penetration of the target host. Thus, the level 0 agent helps open the way for the installation of more complex agents, as discussed below.

[0076] The level 0 agent is often installed directly as the result of exploiting a security vulnerability, e.g., during exploitation of a buffer overflow or user-supplied format string vulnerability in an application running on the target host. The installed agent runs in the process space of the compromised application, e.g., an Internet web server or email server, and opens a socket to provide an unencrypted communication channel to the console or upstream agent. The syscall proxy server **220** in the level 0 agent **205** can execute one syscall at a time.

[0077] **FIG. 6** shows an example of a more complex agent, which is referred to as a level 1 agent **225**. The level 1 agent **225**, like the level 0 agent **205**, provides a syscall proxy server **220** to execute commands received from the console **105** or an upstream agent. The level 1 agent **225** can spawn separate syscall proxy servers **220** for corresponding syscalls received from the upstream agent, so several syscalls can be executed simultaneously. The level 1 agent **225** also can allocate memory and other resources within the host. Hence, the level 1 agent **225** can run in its own process space rather than relying solely on the process space of a compromised application. The ability to run in its own process space gives the level 1 agent **225** more autonomy and control over its own lifetime and more control over the resources of the host **210**.

[0078] The level 1 agent **225** also provides a secure communication channel **230** with authentication and encryption of data. Data transmitted by the level 1 agent **225** back to the console **105** or upstream agent is encrypted to ensure that information about the compromised system cannot be detected by third parties that may be monitoring network communications. The data is authenticated, e.g., using a digital signature, so that only the console that created the agent can communicate with it. This prevents different users who may be running the security compromise system from communicating with each other's agents.

[0079] **FIG. 7** shows an example of a more complex agent, which is referred to as a level 2 agent **305**. The level 2 agent **305**, like the level 1 agent **225**, provides a syscall proxy server **220** and secure communication **230** capability. The syscall proxy server **220** allows the level2 agent to execute syscalls received from a module running on an upstream agent in a manner similar to the level 0 and level 1 agents. For example, a module running on the local agent **135** in the console **105** such as, for example, a remote exploit, may generate system calls that are intended to be executed in the operating system of the target host, rather than the console. Such syscalls are sent to the syscall proxy server **220** of the remote agent in the target host to be executed.

[0080] The level 2 agent **305** also provides a virtual machine **310** that can execute modules written in high-level, platform-independent, scripting languages, such as Python, Perl, Java, and Tcl. A module, in general, is an element of program code containing an individual operation or group of operations that are to be executed. Most of the modules used in the automated penetration test are designed to exploit security vulnerabilities in or gather information from a target host. Such modules may be obtained in a variety of ways.

For example, laboratory testing may be performed on widely used software products to determine possible security vulnerabilities and how to exploit them. A module is then written for each software product based on the results of this testing. As a further example, such modules are commonly written and made available on the Internet by members of the on-line security research community or by hackers or crackers, i.e., persons who attempt to gain unauthorized access to computer systems.

[**0081**] One advantage of executing modules in a virtual machine **310** is that it allows the level 2 agent **305** to perform computationally-intensive operations on the target host **210** without receiving a continuous flow of instructions, as in the case of syscall proxying. Indeed, syscall proxying may not be feasible for computationally-intensive operations because of the delays in transmitting instructions that result from network latency. Thus, the virtual machine **310** of the level 2 agent **305** can take greater advantage of the processor and other resources of the target host **210**.

[**0082**] Another advantage of executing modules in a virtual machine **310** is that the modules need not be included as an integral part of the agent, but instead may be transferred to the agent on demand.

[**0083**] This obviates the need to port the modules to the target host **210** operating system. Thus, modules can be run in a variety of target hosts without being rewritten for each different target host operating system. Moreover, the size of the agent is substantially reduced. In addition, because there are many widely available programs and modules written in standard scripting languages, the amount of custom programming required to create modules to execute on the target host is reduced.

[**0084**] The virtual machine **310** can communicate with, control, and install modules in downstream agents by opening a network connection to the target host **210** to access a network **315**. The communication with the downstream agents may be established by a secure communication module **230** that forms a secure communication channel with data encryption and authentication. The virtual machine **310** can also handle native calls, which are syscalls that are intended to be executed in the operating system of the target host **210** on which the virtual machine **310** is installed. For example, an information gathering module may execute a syscall in the operating system of the target host **210** on which the virtual machine **310** is installed in order to access the Internet through a network interface.

[**0085**] In the example shown in **FIG. 7**, the network **315** accessed by the virtual machine **310** is separate from the network **215** used to communicate with the console **105** or upstream agent. The network **315** accessed by the virtual machine **310** may be a target network **100**, such as shown in **FIGS. 1 and 3**, that cannot be reached from an outside host. Alternatively, as shown in **FIG. 8**, the network **320** accessed by the virtual machine **310** may be the same network **320** used to communicate with the console **105** or upstream agent. For example, as shown in **FIG. 1**, both the second and third target hosts are connected to the target network, and a virtual machine in an agent on the second target host may communicate with an agent installed on the third target host.

[**0086**] The virtual machine **310** and the syscall proxy server **220** in the level 2 agent **305** are controlled by an execution engine **325** that determines which modules are executed and how they are executed. For example, the execution engine **325** provides task control capability to start, pause, resume, and cancel module execution. The execution engine **325** also controls the execution of native modules **330**, which are modules written to be executed directly in the operating system of the target host **210**.

[**0087**] The execution engine **325** provides transparent multitasking capabilities by taking advantage of either the multi-threading or multi-processing capabilities of the underlying operating system to enable concurrent execution of modules and syscalls by spawning new threads or processes as necessary. Task-specific storage is provided to enable an executing module to access context information. For example, upon initialization, the execution engine may set up synchronization objects (known as mutexes) and task-specific storage for module context using, e.g., the ThreadLocalStorage routine in Win32 or the thread-specific routine in PTHREADS.

[**0088**] The execution engine **325** is isolated in certain respects from the rest of the system architecture, aside from the modules that are controlled by it. For example, the execution engine **325** operates independently of the proxycall architecture. In addition, necessary parameters for the modules to execute on the agent are provided by the module context, rather than by the execution engine **325**.

[**0089**] The module context provides all of the information necessary for a specific execution instance (the execution instance being, e.g., a task or executing module). Such information includes the module parameters, designation of remote or local execution, description of the peer (for remote execution), etc. The module context is available from any subsystem of the agent that is involved in the module execution, e.g., a proxycall client. The module accesses the module context from the agent during execution of the module.

[**0090**] The level 2 agent receives commands and instructions through a remote procedure call (RPC) component **327**, which then passes the received data to the execution engine **325**. The RPC component **327** employs a standard protocol to control communication with the console or upstream agent using a predefined data packet format. To implement this protocol, the console uses an internal representation of all the deployed agents that includes all of the properties and capabilities for establishing a communication channels with the deployed agents.

[**0091**] Communications between the console and deployed level 1 and level 2 agents are established as follows. First, the console establishes an internal connection with the internal representation of the remote agent with which it is attempting to communicate. The internal representation of the remote agent uses its communications component to establish a communication channel across the network with the remote level 1 or level 2 agent. The communication channel then is created using a component of the system that can multiplex communications from several modules into one or many communications channels with a remote agent.

[**0092**] The RPC component **327** implements a bidirectional communication mechanism using the communications channel described above as an underlying transport

mechanism. The RPC component implements and executes commands that control the execution of modules on the remote agents as well as the transfer of information between agents. For example, the RPC component is used to start, pause and stop module execution. The RPC component also is used to transfer modules to remote agents, retrieve module execution results from remote agents, and control the configuration and status of remote agents.

[0093] **FIG. 9** shows an example of a level 3 agent **335**, which is similar to the level 2 agent **305** shown in **FIG. 8**, but includes a user interface **340** through which the level 3 agent **335** is controlled. The user interface **340** allows the level 3 agent **335** to function as the local agent **135** component of the console **105**, as shown in **FIG. 2**. The level 3 agent **335**, like the level 2 agent **305**, includes secure communication capability **230**, a virtual machine **310**, a syscall proxy server **220**, native module capability **330**, an RPC send/dispatch component **327**, and an execution engine **325**.

[0094] The proxycall and virtual machine capabilities discussed above enable the agents to execute modules in a number of different configurations. **FIG. 10** shows a configuration in which a module **400** is executed by the local agent **135** in the console **105**. System calls generated by the module **400** are handled by the local agent **135** as native syscalls and are executed in the operating system **405** of the console **105**. For example, when a module **400** executes a command for querying the status of a port on a target host **115**, a system call is made to the operating system **405** of the console **105** in order to access the network interface **140** and transmit the request to the target host **115** through the Internet **110**.

[0095] Another configuration in which a module is executed by the local agent **135** in the console **105** is shown in **FIG. 11**. In this configuration, however, system calls generated by the module **400** are transmitted to a remote agent **120** installed in the first target host **115**. The system calls are handled by the syscall proxy server of the remote agent **120** and are executed in the operating system **410** of the first target host **115**. For example, when a module **400** executes a command for retrieving a file from the second target host **125** through the target network **100**, the syscall generated by this command is executed in the operating system **410** of the first target host **115**. From the perspective of the second target host **125**, the command appears to have been executed on the first target host **115** and therefore is more likely to be trusted and implemented by the second target host **125**.

[0096] **FIG. 12** shows a configuration in which a module **400** is executed by a remote agent **120** in a target host, e.g., the first target host **115**. System calls generated by the module **400** are handled by the remote agent **120** as native syscalls and are executed in the operating system of the first target host **410**. This configuration allows the module **400** to take advantage of the resources of the first target host **115**. In addition, it enables the module **400** to run without generating network traffic between the console **105** and the target network **100**.

[0097] Another configuration in which a module is executed by a remote agent in a target host is shown in **FIG. 13**. In this configuration, however, the system calls generated by the module **400** are transmitted to a remote agent **130** installed in another target host, e.g., the second target host **125**. System calls generated by the module **400** are handled by the syscall proxy server of the remote agent **130** and are executed in the operating system **415** of the second target host **125**.

[0098] As above, this configuration allows the module to take advantage of the resources of the first target host **115**, and enables the module to run without generating network traffic between the console **105** and the target network **100**. Another advantage of this configuration is that, from the perspective of the other hosts in the target network **100**, the module **400** commands appear to originate from the second target host **125**. Consequently, such commands are more likely to be trusted and implemented by the other hosts of the target network. In other words, using the remote agent **130**, the system effectively assumes the identity of the compromised host and takes advantage of the privileges of the compromised host within the target network.

[0099] **FIG. 14** shows another configuration in which a module is executed by the local agent in the console. In this configuration, however, system calls generated by the module **400** are transmitted to a remote agent **120** installed in the first target host **115**, which in turn passes the system calls to a remote agent **130** installed in the second target host **125**. The system calls are handled by the syscall proxy server of the remote agent **130** in the second target host **125** and are executed in the operating system **415** of the second target host **125**. As in the case above, from the perspective of the other hosts in the target network, the module commands appear to originate from the second target host. Consequently, such commands are more likely to be trusted and implemented by the other hosts of the target network.

[0100] As discussed above, the local agent **135** in the console **105** provides a user interface **150**. As shown in **FIG. 15**, the user interface presents to the user a Windows-based graphical user interface (GUI) screen. The screen has several component windows that present information in various formats and allow control to be executed over various aspects of the penetration testing. Of course, the components of the screen may be moved and sized based on the user's preferences.

[0101] The visibility view, in the central portion of the screen, presents a graphical representation, i.e., a model, of the target network. In this example, the graphical representation is in the form of a hierarchical listing of entities, i.e., a tree listing. Each entity represents a physical component of the target network, e.g., a host. Initially, the visibility view shows only two entities, the local host (e.g., the console) and the local agent that is running on the local host. As further discussed below, as each stage of the penetration testing is performed, the model of the target network is updated to reflect newly gained information.

[0102] Tabs are arranged on the left side of the screen to organize the available functions, modules, and tools offered by the program. Clicking on a tab causes it to move to the top of the window and display the available modules under that category. For example, the Remote Exploits tab provides various modules that may be run on the local agent to exploit security vulnerabilities in a target network without being located in the target network. An executed modules window is provided on the top right side of the screen to display a history of the modules that have been executed

during the penetration testing and their status. A module output window is provided on the bottom right side of the screen to display the output of the modules.

[0103] **FIG. 16** shows an example of how the user interface screen appears after the Network Discovery module has been run. The Network Discovery module, which is grouped under the Information Gathering tab, is typically run early in the penetration testing to determine the topology of the target network. The executed modules window shows that the Network Discovery module has been run and provides the start and finish times and the current status, which in this case indicates that the execution of the module is finished. The module output window provides a table of host names, Internet protocol (IP) addresses, operating system (OS), and other information output by the module.

[0104] Following the execution of the Network Discovery module, the visibility view displays a graphical representation of the target network in the form of a tree listing. The tree shows that the local host on the console (local host) has a local agent that is connected to a first target host having IP address 192.168.66.0/24. The first target host, in turn, is connected to other hosts within the target network. The entities in the tree listing may be collapsed or expanded according to hierarchical level by clicking on the arrow symbols on the left-hand side of the entities. As each module is run during the penetration testing, the visibility view is automatically updated to show a more complete model of the target network.

[0105] **FIG. 17** shows an example of the screen display for initiating the execution of the Network Discovery module for a second time. The user double clicks on the Network Discovery module tab, and the screen presents a module parameter entry box. The user is prompted to enter the IP address of the target network, which is typically known by the user prior to performing penetration testing. For instance, the user may determine the IP address from a domain name server lookup of the domain name of the target host. In this example, the Network Discovery module is targeting the host with IP address 192.168.22.0/24. In the visibility view, the previously targeted host (192.168.66.0/24) has been collapsed to a single top-level entry in the tree listing. As an alternative to entering the target host IP address, the module tab may be dragged and dropped onto the target host in the tree listing for which information is to be gathered.

[0106] As discussed above, modules may be run on local or remote agents in a variety of configurations. The system calls generated by the modules are executed on the default source host, which in the example of **FIG. 17** is the local agent. The default source is either the most recently used source or a source selected in the visibility view prior to execution of the module. For instance, if a module is executed using the local agent as a source host, the local agent remains the default source until a new source is selected. Thus, double clicking on the Network Discovery tab results in that module being run with on the local agent and gathering information from the target host entered in the module parameter entry box.

[0107] As shown in **FIG. 18**, the output of the Network Discovery module is presented in the visibility view as a list of connected hosts under the target host (192.168.22.0/24). The executed module window indicates that the Network Discovery module has been run a second time. In addition, the Remote Procedure Call (RPC) Mapper module has been run, which attempts to identify all of the ports in the target host that are available to receive RPC commands. The RPC Mapper was initiated by dragging and dropping it onto the target host, which in this case is 192.168.22.2. The output of the RPC Mapper is shown in the module output window in the log/debug format, which includes information to enable the user to track and rectify errors in the module. The output identifies the RPC ports and their associated protocols.

[0108] As shown in **FIG. 19**, the entities in the model of the target network can be examined and modified using an entity editor. Each entity has properties that reflect aspects of the information gathered on the corresponding physical component in the target network. For example, a host entity has properties such as system architecture (arch), operating system, interfaces, ports, etc. The host entity also has an agent property that lists the agents running on that host.

[0109] The properties of each entity are updated automatically as modules are run during the penetration testing. For example, the entity corresponding to host 192.168.22.2 has a listing of ports available for RPC service. This listing is the result of running the RPC Mapper module directed toward this host, as described above. In certain instances, different modules may result in different property states, for example, two different results regarding the operating system being run on a particular host. In such a case, the program will prompt the user to resolve the apparent conflict based on experience or outside knowledge.

[0110] Even in the absence of a conflict, the state of these properties may be changed by the user based on experience and knowledge gained from other sources. For example, the user may know, based on experience, that a host having a particular architecture, e.g., Sparc Version 8, may run a particular operating system, e.g., Solaris. In such a case, the user edits the host entity and changes the operating system property from unknown to Solaris. This allows the user to build a more complete and accurate model of the target network.

[0111] Once sufficient information has been gathered regarding the target network, modules may be executed to exploit security vulnerabilities in the target network. As shown in **FIG. 20**, such modules are grouped under the Local Exploits and Remote Exploits tabs on the user interface screen depending upon whether they run within the target host (i.e., locally) or remotely. In this example, the executed module window shows that a module called "General Exploit" has been run twice. The first execution of the module was unsuccessful, as indicated by the icon to the left of the module name and by the term "aborted" in the status column. The second execution was successfully completed. In addition to the modules grouped under the exploit tabs, there are a series of modules under the Agents tab that relate to the installation and execution of remote agents in the target network. In this example, a level 0 agent has been installed in the host with address 192.168.22.6.

[0112] The installation of the level 0 agent may be accomplished by clicking on two entries under the Agents tab. First, a level 0 agent service is initiated, which is a process that runs on the target host. As discussed above, the level 0 agent service may share the process space of an existing service on the target host, such as an email server. Then, a

level 0 agent is installed and executed in the level 0 agent service. Alternatively, the program may present a single entry under the Agents tab that performs all of the steps required to install and execute the level 0 agent.

[0113] Once the level 0 agent has been installed, the user can begin to take advantage of the agent by executing modules on it. The user sets the level 0 agent to be the source for the execution of modules by right clicking on the agent and selecting "Set as source". As shown in **FIG. 21**, the level 0 agent may be displayed as highlighted or bold in the visibility view to indicate to the user that it is the current default source. All subsequent modules are run on the level 0 agent source until the default is changed.

[0114] In this example, the Python Console module has been initiated. As discussed above, Python is a high-level, platform-independent, scripting language. The Python console allows the user to enter and execute Python commands and to save commands as a module. The Python command interpreter runs on the local agent, however, the syscalls generated by the command interpreter are executed by the syscall proxy server of the level 0 agent, which is the selected source.

[0115] In contrast, a module may run on a remote agent that has the capability to execute modules, such as a level 2 agent. As discussed above, the level 2 agent has a virtual machine that enables it to execute scripting language modules. In such a case, the module is resident on and executes on the remote agent. Syscalls generated by the module are executed either as a native call on the remote host (see **FIGS. 7 and 8**) or by a syscall proxy server on a downstream agent. The module may be installed together with the remote agent, or may be downloaded after the remote agent has been installed. This capability for 100% remote execution is advantageous, because it allows the module to run without receiving a continuous series of commands, thereby avoiding the generation of excessive network traffic. Moreover, a module for exploiting a security vulnerability may have certain timing requirements that, due to network latency, cannot be met if commands must be received over the network.

[0116] Regardless of whether a module is being run on the local agent or a remote agent, the syscalls generated by the module will be executed on the selected source, which in the example of **FIG. 21** is the level 0 agent installed on target host 192.168.22.6. Thus, the Python commands executed in the Python console generate syscalls that are executed on this selected target host. For example, the Python command "print socket.gethostname ( )", shown in the Python console window, generates a syscall to retrieve the name of the target host, "diavolo". The remaining commands shown in the window generate syscalls to create a socket in the target host, which can he be used to retrieve information from the target host.

[0117] It will be appreciated that each of these embodiments discussed above provides a novel system and method for analyzing computer system security by compromising the security in a systematic manner.

[0118] It also will be appreciated that because the system provides a database of modules for exploiting security vulnerabilities, the system described herein actually attempts to exploit detected vulnerabilities, rather than merely listing them.

[0119] It also will be appreciated that because the system provides a virtual machine to execute modules in agents installed in target hosts, the modules can be run on a variety of platforms and operating systems without further customization and testing. In addition, publicly available programs to exploit security vulnerabilities can be used without an extensive laboratory infrastructure for rewriting and testing.

[0120] It also will be appreciated that because the system maintains a database of all operations performed during the penetration testing, the target system can be reliably returned to its original configuration.

[0121] While the present invention has been described with respect to what is presently considered to be the preferred embodiments, it is to be understood that the invention is not limited to the disclosed embodiments. To the contrary, the invention is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

What is claimed:

1. A system for performing penetration testing of a target computer network by installing a remote agent in the target computer network, the system comprising:

a local agent provided in a console and configured to receive and execute commands;

a user interface provided in the console and configured to send commands to and receive information from the local agent, process the information, and present the processed information;

a database configured to store the information received from the local agent;

a network interface connected to the local agent and configured to communicate via a network with the remote agent installed in the target computer network; and

security vulnerability exploitation modules for execution by the local agent and/or the remote agent.

2. The system of claim 1, wherein the user interface enables a user to select one of the modules and initiate execution of the selected module on either the local agent or the remote agent.

3. The system of claim 1, wherein the user interface provides a graphical representation of the target computer network.

4. A method for performing penetration testing of a target computer network, comprising:

installing a remote agent in the target computer network;

executing a command using a local agent provided in a console;

receiving information from the local agent in a user interface provided in the console;

presenting the information received from the local agent to a user;

storing the information received from the local agent in a database;

communicating via a network with the remote agent installed in the target computer network; and

providing security vulnerability exploitation modules for execution by the local agent and/or the remote agent.

5. The method of claim 4, further comprising:

selecting, using the user interface, one of the modules; and

initiating execution of the selected module on either the local agent or the remote agent.

6. The method of claim 4, further comprising providing a graphical representation of the target computer network using the user interface.

7. Computer code for performing penetration testing of a target computer network, the computer code comprising code for:

installing a remote agent in the target computer network;

executing a command using a local agent provided in a console;

receiving information from the local agent in a user interface provided in the console;

presenting the information received from the local agent to a user;

storing the information received from the local agent in a database;

communicating via a network with the remote agent installed in the target computer network; and

providing security vulnerability exploitation modules for execution by the local agent and/or the remote agent.

8. The computer code of claim 7, further comprising code for:

selecting, using the user interface, one of the modules; and

initiating execution of the selected module on either the local agent or the remote agent.

9. The computer code of claim 7, further comprising code for providing a graphical representation of the target computer network using the user interface.

10. An agent for use in a system for performing penetration testing of a target computer network, the agent comprising:

a proxy server configured to receive and execute system calls received via a network; and

a virtual machine configured to execute scripting language instructions received via the network.

11. The agent of claim 10, further comprising an execution engine configured to control the proxy server and the virtual machine, wherein the system calls and the scripting language instructions are routed to the proxy server and the virtual machine, respectively, by the execution engine.

12. The agent of claim 11, further comprising a remote procedure call module configured to receive commands from the network formatted in a remote procedure call protocol and pass the commands to the execution engine.

13. An agent for use in a system for performing penetration testing of a target computer network, the agent comprising:

a proxy server configured to receive and execute system calls received via a network;

a virtual machine configured to execute scripting language instructions received via the network;

a secure communication module configured to provide secure communication between the virtual machine and the network;

an execution engine configured to control the proxy server and the virtual machine, wherein the system calls and the scripting language instructions are routed to the proxy server and the virtual machine, respectively, by the execution engine;

a remote procedure call module configured to receive commands via the network formatted in a remote procedure call protocol and pass the commands to the execution engine; and

a second secure communication module configured to provide secure communication between the remote procedure call module and the network.

14. A method for performing penetration testing of a target network, comprising the steps of:

executing a first module in a console having a user interface, the first module being configured to exploit a security vulnerability in a first target host of the target network;

installing a first remote agent in the first target host, the first remote agent being configured to communicate with the console and a second remote agent; and

executing a second module in the first remote agent, the second module being configured to exploit a security vulnerability in a second target host of the target network.

15. The method of claim 14, further comprising installing a second remote agent in the second target host of the target network, the second remote agent being configured to communicate with the first remote agent.

16. A system for performing penetration testing of a target network, comprising:

a console having a user interface;

a first module configured to execute in the console to exploit a security vulnerability in a first target host of the target network;

a first remote agent installed in the first target host, the first remote agent being configured to communicate with the console and a second remote agent; and

a second module configured to execute in the first remote agent to exploit a security vulnerability in a second target host of the target network.

17. The system of claim 16, further comprising a second remote agent installed in the second target host of the target network, the second remote agent being configured to communicate with the first remote agent.

18. Computer code for performing penetration testing of a target network, the computer code comprising code for:

executing a first module in a console having a user interface, the first module being configured to exploit a security vulnerability in a first target host of the target network;

installing a first remote agent in the first target host, the first remote agent being configured to communicate with the console and a second remote agent; and

executing a second module in the first remote agent, the second module being configured to exploit a security vulnerability in a second target host of the target network.

19. The computer code of claim 18, further comprising code for installing a second remote agent in the second target host of the target network, the second remote agent being configured to communicate with the first remote agent.

20. A method for performing penetration testing of a target network, comprising the steps of:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

sending a system call to the first remote agent via a network; and

executing the system call in the first target host using a proxycall server of the first remote agent to exploit a security vulnerability of a second target host.

21. A method for performing penetration testing of a target network, comprising the steps of:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

executing in the first remote agent a second module that generates a system call; and

executing the system call in the first target host to exploit a security vulnerability of a second target host.

22. A method for performing penetration testing of a target network, comprising the steps of:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

executing a second module in the first remote agent that generates a system call;

installing a second remote agent in the second target host as a result of exploiting a security vulnerability of the second target host;

sending the system call generated by the second module to the second remote agent via a network; and

executing the system call in the second target host using a proxycall server of the second remote agent.

23. A method for performing penetration testing of a target network, comprising the steps of:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

installing a second remote agent in the second target host as a result of exploiting a security vulnerability of the second target host;

sending a system call to the first remote agent;

sending the system call from the first remote agent to the second remote agent; and

executing the system call in the second target host using a proxycall server of the second remote agent.

24. A method for performing penetration testing of a target network, comprising the steps of:

installing a first remote agent in the first target host, the first remote agent having a proxy server configured to receive and execute system calls;

executing in the first remote agent a system call received via a network;

installing a second remote agent in the first target host, the second remote agent having a proxy server configured to receive and execute system calls and a virtual machine configured to execute scripting language instructions; and

executing in the second remote agent a scripting language instruction or a system call, the system call being received via the network.

25. Computer code for performing penetration testing of a target network, the code comprising code for:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

sending a system call to the first remote agent via a network; and

executing the system call in the first target host using a proxycall server of the first remote agent to exploit a security vulnerability of a second target host.

26. Computer code for performing penetration testing of a target network, the code comprising code for:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

executing in the first remote agent a second module that generates a system call; and

executing the system call in the first target host to exploit a security vulnerability of a second target host.

27. Computer code for performing penetration testing of a target network, the code comprising code for:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

executing a second module in the first remote agent that generates a system call;

installing a second remote agent in the second target host as a result of exploiting a security vulnerability of the second target host;

sending the system call generated by the second module to the second remote agent via a network; and

executing the system call in the second target host using a proxycall server of the second remote agent.

28. Computer code for performing penetration testing of a target network, the code comprising code for:

executing a first module to exploit a security vulnerability of a first target host of the target network;

installing a first remote agent in the first target host as a result of exploiting the security vulnerability of the first target host;

installing a second remote agent in the second target host as a result of exploiting a security vulnerability of the second target host;

sending a system call to the first remote agent;

sending the system call from the first remote agent to the second remote agent; and

executing the system call in the second target host using a proxycall server of the second remote agent.

29. Computer code for performing penetration testing of a target network, the code comprising code for:

installing a first remote agent in the first target host, the first remote agent having a proxy server configured to receive and execute system calls;

executing in the first remote agent a system call received via a network;

installing a second remote agent in the first target host, the second remote agent having a proxy server configured to receive and execute system calls and a virtual machine configured to execute scripting language instructions; and

executing in the second remote agent a scripting language instruction or a system call, the system call being received via the network.

* * * * *